

# Package ‘transmogR’

January 20, 2026

**Type** Package

**Title** Modify a set of reference sequences using a set of variants

**Version** 1.6.0

**Description** transmogR provides the tools needed to create a new reference genome or reference transcriptome, using a set of variants. Variants can be any combination of SNPs, Insertions and Deletions. The intended use-case is to enable creation of variant-modified reference transcriptomes for incorporation into transcriptomic pseudo-alignment workflows, such as salmon.

**License** GPL-3

**Encoding** UTF-8

**URL** <https://github.com/smped/transmogR>

**BugReports** <https://github.com/smped/transmogR/issues>

**Depends** R (>= 4.1.0), Biostrings, GenomicRanges

**Imports** BSgenome, data.table, Seqinfo, GenomicFeatures, ggplot2 (>= 4.0.0), IRanges, jsonlite, matrixStats, methods, parallel, patchwork, scales, stats, S4Vectors, SummarizedExperiment, VariantAnnotation

**Suggests** BiocStyle, BSgenome.Hsapiens.UCSC.hg38, edgeR, extraChIPs, InteractionSet, knitr, readr, rmarkdown, rtracklayer, SimpleUpset, testthat (>= 3.0.0)

**biocViews** Alignment, GenomicVariation, Sequencing, TranscriptomeVariant, VariantAnnotation

**BiocType** Software

**VignetteBuilder** knitr

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.3

**Config/testthat/edition** 3

**git\_url** <https://git.bioconductor.org/packages/transmogR>

**git\_branch** RELEASE\_3\_22

**git\_last\_commit** 7065211

**git\_last\_commit\_date** 2025-10-29

**Repository** Bioconductor 3.22

**Date/Publication** 2026-01-19  
**Author** Stevie Pederson [aut, cre] (ORCID:  
    <https://orcid.org/0000-0001-8197-3303>)  
**Maintainer** Stevie Pederson <stephen.pederson.au@gmail.com>

Contents

transmogR-package . . . . .	2
cleanVariants . . . . .	3
digestSalmon . . . . .	4
genomogrify . . . . .	6
indelcator . . . . .	9
overlapsByVar . . . . .	10
owl . . . . .	11
parY . . . . .	12
shiftByVar . . . . .	13
sjFromExons . . . . .	14
transmogrify . . . . .	16
upsetVarByCol . . . . .	18
varTags . . . . .	20
varTypes . . . . .	21
<b>Index</b>	<b>23</b>

---

transmogR-package	<i>transmogR: Create a variant-modified reference transcriptome</i>
-------------------	---

---

Description

The package transmogR has been designed for creation of a variant-modified reference transcriptome

Details

The package transmogR provides two primary functions for modifying complete transcriptomes or genomes:

- [transmogrify\(\)](#) for incorporating the supplied variants into transcriptomic sequences, and
- [genomogrify\(\)](#) for incorporating the supplied variants into genomic sequences, ideally to be passed as decoy sequences to a tool such as salmon.

The main functions rely on lower-level functions such as:

- [owl\(\)](#) which over-writes letters (i.e. SNPs) within a sequence, and
- [indelcator\(\)](#) which incorporates InDels into an individual sequence

Additional utility functions are provided which allow characterisation and exploration of any set of variants:

- [overlapsByVar\(\)](#) counts the variants which overlap sets of GenomicRanges, first splitting the variants into SNV, Insertions and Deletions

- `parY()` returns the pseudo-autosomal regions for a chosen genome build as a `GenomicRanges` object
- `upsetVarByCol()` produces an UpSet plot counting how many unique IDs are impacted by a set of variants. IDs can represent any column in the supplied ranges, such as `gene_id` or `transcript_id`
- `varTypes()` classifies a set of variants into SNV, Insertions of Deletions

### Author(s)

Stevie Pederson

### See Also

Useful links:

- <https://github.com/smped/transmogR>
- Report bugs at <https://github.com/smped/transmogR/issues>

---

cleanVariants

*Check provided variants for compatibility*

---

### Description

Check and cleanup a set of variants for downstream compatibility

### Usage

```
cleanVariants(var, ...)

## S4 method for signature 'GRanges'
cleanVariants(var, ol_vars = "fail", ref_col = "REF", alt_col = "ALT", ...)

## S4 method for signature 'VcfFile'
cleanVariants(
  var,
  ol_vars = "fail",
  which,
  ref_col = "REF",
  alt_col = "ALT",
  ...
)
```

### Arguments

<code>var</code>	GRanges object containing the variants, or a <a href="#">VariantAnnotation::VcfFile</a>
<code>...</code>	Not used
<code>ol_vars</code>	Error handling for any overlapping variants. Can take values in <code>c("fail", "none", "first", "last", "longest", "shortest")</code> . Default is set to fail, with additional options to drop all overlapping variants ('none'), select by genomic position ('first', 'last'), or select by the scale of change to the genome ('longest', 'shortest')

ref\_col, alt\_col  
 Column names corresponding to the reference and alternate alleles

which  
 Passed to [VariantAnnotation::ScanVcfParam](#) if working with a VcfFile

### Details

This function checks a set of variants for the expected structure which is required by all downstream functions in transmogR. The primary change to the data structure is that both REF and ALT columns will be set as simple character vectors.

Given the complicated variant calls that can often be produced by variant callers, additional checks performed will be to ensure that:

- there are no overlapping variants
- SNPs are all single nucleotides and not longer strings
- SNPs are bi-allelic
- Insertions contain a single nucleotide in the REF column
- Deletions contain a single nucleotide in the ALT column
- No missing values are present
- All ALT/REF nucleotides conform to IUPAC codings

### Value

GRanges object with any incompatible variants removed, or an error produced. The mcols will contain the columns REF and ALT, unless otherwise specified, as character vectors

### Examples

```
# Any conflicting variants will be removed
var <- GRanges(c("chr10:114468420-114468422", "chr10:114468422"))
var$REF <- c("GCC", "C")
var$ALT <- c("G", "CTAT")
var
## Taken from the 1000GP, the first variant would delete the C at 114468422
## whilst the second variant begins an insertion at this position.
## These are clearly conflicting. The default value for ol_vars is to fail
## with an error (ol_vars = "fail"). However, both can be removed by setting
## ol_vars = "none". A warning will always be produced.
cleanVariants(var, ol_vars = "none")
## Or the longest can be retained, along with multiple other options
cleanVariants(var, ol_vars = "longest")
```

---

digestSalmon

*Parse the output from salmon*

---

### Description

Parse transcript counts and additional data from salmon

Calculate the overdispersions from a set of paths without parsing any counts

**Usage**

```
digestSalmon(
  paths,
  max_sets = 2L,
  aux_dir = "aux_info",
  name_fun = basename,
  verbose = TRUE,
  extra_assays = NULL,
  max_boot = Inf,
  ...
)

overdispFromBoots(paths, n_boot, .ids)
```

**Arguments**

<code>paths</code>	Vector of file paths to directories containing salmon results
<code>max_sets</code>	The maximum number of indexes permitted
<code>aux_dir</code>	Subdirectory where bootstraps and meta_info.json are stored
<code>name_fun</code>	Function applied to paths to provide colnames in the returned object. Set to <code>NULL</code> or <code>c()</code> to disable.
<code>verbose</code>	Print progress messages
<code>extra_assays</code>	Can take values in <code>c("TPM", "effectiveLength", "length")</code> to optionally request TPM, effectiveLength or length as assays. Including the length assay is intended for the use case of personalised transcriptomes where transcript lengths may no longer be uniform across samples. None will be returned by default
<code>max_boot</code>	The maximum number of bootstraps to use. Setting this to zero will ignore all bootstraps and the scaledCounts assay will not be included in the returned object
<code>...</code>	Not used
<code>n_boot</code>	The number of bootstraps
<code>.ids</code>	Vector of transcript IDs which match the bootstrap values. Will be parsed from paths if not provided, although this adds time

**Details**

This function is based heavily on `edgeR::catchSalmon()` however, there are some important differences:

1. A `SummarizedExperiment` object is returned
2. Differing numbers of transcripts are allowed between samples

The second point is intended for the scenario where some samples may have been aligned to a full reference, with remaining samples aligned to a partially masked reference (e.g. chrY). This will lead to differing numbers of transcripts within each salmon index, however, common estimates of overdispersions are required for scaling transcript-level counts. By default, the function will error if >2 different sets of transcripts are detected, however this can be modified using the `max_sets` argument.

This greater flexibility also requires more stringent checking and, as such, for smaller datasets, `digestSalmon` may be slower than the `edgeR` function.

The SummarizedExperiment object returned may also contain multiple assays, as described elsewhere on this page

This follows the methods of Baldoni, et al. (2024). Dividing out quantification uncertainty allows efficient assessment of differential transcript expression with edgeR. Nucleic Acids Research, 52(3), e13. <https://doi.org/10.1093/nar/gkad1167>

### Value

A SummarizedExperiment object containing assays for counts and scaledCounts. The scaledCounts assay contains counts divided by overdispersions. rowData in the returned object will also include transcript-lengths along with the overdispersion estimates used to return the scaled counts. TPM, effectiveLength and length can be returned as additional assays by specifying one or more of these in the extra\_assays argument

overdispFromBoots returns a numeric vector

### Examples

```
## Provide the path to the parent directories which contains each
## quant.sf file
ex_path <- system.file("extdata/salmon_test", package = "transmogR")
se <- digestSalmon(ex_path, extra_assays = "TPM", verbose = FALSE)
se

ex_path <- system.file("extdata/salmon_test", package = "transmogR")
overdispFromBoots(ex_path, 10)
```

---

genomogrify

*Mogrify a genome using a set of variants*

---

### Description

Use a set of SNPS, insertions and deletions to modify a reference genome

### Usage

```
genomogrify(x, var, ...)

## S4 method for signature 'XStringSet,GRanges'
genomogrify(
  x,
  var,
  alt_col = "ALT",
  mask = GRanges(),
  tag = NULL,
  sep = "_",
  var_tags = FALSE,
  var_sep = "_",
  ol_vars = "fail",
  verbose = TRUE,
  ...)
```

```
)

## S4 method for signature 'BSgenome,GRanges'
genomogrify(
  x,
  var,
  alt_col = "ALT",
  mask = GRanges(),
  names,
  tag = NULL,
  sep = "_",
  var_tags = FALSE,
  var_sep = "_",
  ol_vars = "fail",
  verbose = TRUE,
  ...
)

## S4 method for signature 'BSgenome,VcfFile'
genomogrify(
  x,
  var,
  alt_col = "ALT",
  mask = GRanges(),
  names,
  tag = NULL,
  sep = "_",
  var_tags = FALSE,
  var_sep = "_",
  ol_vars = "fail",
  which,
  verbose = TRUE,
  ...
)

## S4 method for signature 'XStringSet,VcfFile'
genomogrify(
  x,
  var,
  alt_col = "ALT",
  mask = GRanges(),
  tag = NULL,
  sep = "_",
  var_tags = FALSE,
  var_sep = "_",
  ol_vars = "fail",
  which,
  verbose = TRUE,
  ...
)
```

## Arguments

<code>x</code>	A DNASTringSet or BSgenome
<code>var</code>	GRanges object containing the variants, or a <a href="#">VariantAnnotation::VcfFile</a>
<code>...</code>	Passed to <a href="#">parallel::mclapply</a>
<code>alt_col</code>	The name of the column with var containing alternate bases
<code>mask</code>	Optional GRanges object defining regions to be masked with an 'N'
<code>tag</code>	Optional tag to add to all sequence names which were modified
<code>sep</code>	Separator to place between seqnames names & tag
<code>var_tags</code>	logical(1) Add tags indicating which type of variant were incorporated, with 's', 'i' and 'd' representing SNPs, Insertions and Deletions respectively
<code>var_sep</code>	Separator between any previous tags and variant tags
<code>ol_vars</code>	Error handling for any overlapping variants. See <a href="#">cleanVariants</a> for possible values and an explanation
<code>verbose</code>	logical(1) Print progress messages while running
<code>names</code>	Sequence names to be mogrified
<code>which</code>	GRanges object passed to <a href="#">VariantAnnotation::ScanVcfParam</a> if using a VCF directly

## Details

This function is designed to create a variant-modified reference genome, intended to be included as a set of decoys when using salmon in selective alignment mode. Sequence lengths will change if InDels are included and any coordinate-based information will be lost on the output of this function.

Tags are able to be added to any modified sequence to assist identifying any changes that have been made to a sequence.

## Value

XStringSet with variant modified sequences

## Examples

```
library(GenomicRanges)
dna <- DNASTringSet(c(chr1 = "ACGT", chr2 = "AATTT"))
var <- GRanges(c("chr1:1", "chr1:3", "chr2:1-3"))
var$ALT <- c("C", "GG", "A")
dna
genomogrify(dna, var)
genomogrify(dna, var, tag = "mod")
genomogrify(dna, var, var_tags = TRUE)
genomogrify(dna, var, mask = GRanges("chr2:1-5"), var_tags = TRUE)
```



---

indelcator	<i>Substitute InDels into one or more sequences</i>
------------	---

---

## Description

Modify one or more sequences to include Insertions or Deletions

## Usage

```
indelcator(x, indels, ...)

## S4 method for signature 'XString,GRanges'
indelcator(x, indels, exons, alt_col = "ALT", ol_vars = "fail", ...)

## S4 method for signature 'DNAStringSet,GRanges'
indelcator(
  x,
  indels,
  alt_col = "ALT",
  ol_vars = "fail",
  mc.cores = 1,
  verbose = TRUE,
  ...
)

## S4 method for signature 'BSgenome,GRanges'
indelcator(
  x,
  indels,
  alt_col = "ALT",
  ol_vars = "fail",
  mc.cores = 1,
  names,
  ...
)
```

## Arguments

x	Sequence of class XString
indels	GRanges object with InDel locations and the alternate allele
...	Passed to <a href="#">parallel::mclapply</a>
exons	GRanges object containing exon structure for x
alt_col	Column containing the alternate allele
ol_vars	Error handling for any overlapping variants. See <a href="#">cleanVariants</a> for possible values and an explanation
mc.cores	Number of cores to use when calling <a href="#">parallel::mclapply</a> internally
verbose	logical(1) Print all messages
names	passed to BSgenome::getSeq() when x is a BSgenome object

Details

This is a lower-level function relied on by both `transmogrify()` and `genomogrify()`.  
Takes an `Biostrings::XString` or `Biostrings::XStringSet` object and modifies the sequence to incorporate InDels. The expected types of data determine the behaviour, with the following expectations describing how the function will incorporate data

Input Data Type	Exons Required	Use Case	Returned
XString	Y	Modify a Reference Transcriptome	XString
DNAStringSet	N	Modify a Reference Genome	DNAStringSet
BSgenome	N	Modify a Reference Genome	DNAStringSet

Value

A DNAStringSet or XString object (See Details)

See Also

`transmogrify()` `genomogrify()`

Examples

```
## Start with a DNAStringSet
library(GenomicRanges)
seq <- DNAStringSet(c(seq1 = "AATCTGCGC"))
## Define an Insertion
var <- GRanges("seq1:1")
var$ALT <- "AAA"
seq
indelcator(seq, var)

## To modify a single transcript
library(GenomicFeatures)
ex <- GRanges(c("seq1:1-3:+", "seq1:7-9:+"))
orig <- extractTranscriptSeqs(seq, GRangesList(tx1 = ex))["tx1"]
orig
indelcator(orig, var, exons = ex)
```

---

overlapsByVar	<i>Count overlaps by variant type</i>
---------------	---------------------------------------

---

Description

Count how many variants of each type overlap ranges

**Usage**

```
overlapsByVar(x, var, ...)

## S4 method for signature 'GRangesList,GRanges'
overlapsByVar(x, var, alt_col = "ALT", ...)

## S4 method for signature 'GRanges,GRanges'
overlapsByVar(x, var, alt_col = "ALT", ...)
```

**Arguments**

x	A GRangesList with features of interest
var	A Granges object with variants of interest
...	Passed to <a href="#">rowSums</a>
alt_col	The column within mcols(var) which contains the alternate allele

**Details**

Taking any GRanges or GRangesList, count how many of each variant type overlap a region.

**Value**

A vector or matrix

**Examples**

```
library(rtracklayer)
library(VariantAnnotation)
gtf <- import.gff(
  system.file("extdata/gencode.v44.subset.gtf.gz", package = "transmogR")
)
grl <- splitAsList(gtf, gtf$type)
vcf <- system.file("extdata/1000GP_subset.vcf.gz", package = "transmogR")
var <- rowRanges(readVcf(vcf, param = ScanVcfParam(fixed = "ALT")))
overlapsByVar(grl, var)
```

owl

*OverWrite Letters in an XStringSet***Description**

OverWrite Letters (e.g. SNPs) in an XStringSet

**Usage**

```
owl(seq, snps, ...)

## S4 method for signature 'XStringSet,GRanges'
owl(seq, snps, alt_col = "ALT", ol_vars = "fail", ...)

## S4 method for signature 'BSgenome,GRanges'
owl(seq, snps, alt_col = "ALT", names, ...)
```

**Arguments**

seq	A BSgenome, DNASTringSet, RNASTringSet or other XStringSet.
snps	A GRanges object with SNP positions and a column containing the alternate allele
...	Passed to <code>Biostrings::replaceLetterAt()</code>
alt_col	Column name in the mcols element of snps containing the alternate allele
ol_vars	Error handling for any overlapping variants. See <a href="#">cleanVariants</a> for possible values and an explanation
names	Sequence names to operate on

**Details**

This is a lower-level function called by `transmogrify()` and `genomogrify()`, but able to be called by the user if needed

Note that when providing a BSgenome object, this will first be coerced to a DNASTringSet which can be time consuming.

**Value**

An object of the same class as the original object, but with SNPs inserted at the supplied positions

**Examples**

```
seq <- DNASTringSet(c(chr1 = "AAGC"))
snps <- GRanges("chr1:2")
snps$ALT <- "G"
snps
seq
owl(seq, snps)
```

---

parY

---

*Get the PAR-Y Regions From a Seqinfo Object*


---

**Description**

Define the Pseudo-Autosomal Regions from a Seqinfo Object

**Usage**

```
parY(x, ...)

## S4 method for signature 'Seqinfo'
parY(x, ...)

## S4 method for signature 'character'
parY(x, prefix = NULL, ...)
```

**Arguments**

x	A Seqinfo object or any of named build. If passing a character vector, <code>match.arg()</code> will be used to match the build.
...	Not used
prefix	Optional prefix to place before chromosome names. Can only be NULL, "" or "chr"

**Details**

Using a seqinfo object based on either hg38, hg19, CHM13.v2 or their variations, create a GRanges object with the Pseudo-Autosomal Regions from the Y chromosome for that build. The length of the Y chromosome on the seqinfo object is used to determine the correct genome build when passing a Seqinfo object. Otherwise

An additional mcols column called PAR will indicate PAR1 and PAR2

**Value**

A GenomicRanges object

**Examples**

```
library(Seqinfo)
sq <- Seqinfo(
  seqnames = "chrY", seqlengths = 59373566, genome = "hg19_only_chrY"
)
parY(sq)

## PAR regions for CHM13 are also available
sq <- Seqinfo(
  seqnames = "chrY", seqlengths = 62460029, genome = "CHM13"
)
parY(sq)

## Or just call by name
parY("GRCh38", prefix = "chr")
```

---

shiftByVar

---

*Calculate new exon co-ordinates*


---

**Description**

Calculate new exon co-ordinates after including InDels

**Usage**

```
shiftByVar(x, var, alt_col = "ALT", mc.cores = 1, ...)
```

**Arguments**

<code>x</code>	A <code>GenomicRanges</code> object with co-ordinates needing to be recalculated
<code>var</code>	A set of variants to be incorporated into a reference genome
<code>alt_col</code>	The name of the column with the alternate sequence for each variant
<code>mc.cores</code>	Passed internally to <code>parallel::mclapply()</code>
<code>...</code>	Not used

**Details**

Given a set of variants, this will return a set of genomic ranges with updated co-ordinates able to be applied on a variant modified reference genome

**Value**

`GRanges` object with co-ordinates shifted according to the provided variants. The new co-ordinates will be compatible with a variant-modified genome as produced by `genomogrify()` and can be used to extract the sequences associated with the ranges in the modified reference.

**Examples**

```
# Define a 3nt insertion
var <- GRanges("seq1:5:*", seqlengths = c(seq1=10), REF = "A", ALT = "AGT")
var
# A simple GRanges to shift co-ordinates for
gr <- GRanges("seq1:1-10:+", seqlengths = c(seq1=10), feature = "feature1")
gr
# Create shifted co-ordinates based on the provided variants
new_gr <- shiftByVar(gr, var)
new_gr
## The seqlengths will have been adjusted to account for all variants
seqinfo(new_gr)
```

---

sjFromExons

---

*Obtain Splice-Junctions from Exons and Transcripts*


---

**Description**

Using `GRanges` defining exons and transcripts, find the splice-junctions

**Usage**

```
sjFromExons(
  x,
  rank_col = c("exon_number", "exon_rank"),
  tx_col = c("transcript_id", "tx_id"),
  extra_cols = "all",
  don_len = 8,
  acc_len = 5,
  as = c("GRanges", "GInteractions"),
  ...
)
```

**Arguments**

<code>x</code>	GRanges object with exons and transcripts. A column indicating the position (or rank) of each exon within the transcript must be included.
<code>rank_col</code>	The column containing the position of each exons within the transcript
<code>tx_col</code>	The column containing unique transcript-level identifiers
<code>extra_cols</code>	Can be a vector of column names to return beyond <code>rank_col</code> and <code>tx_col</code> . By default all columns are returned ( <code>extra_cols = "all"</code> ).
<code>don_len, acc_len</code>	Length of donor and acceptor sites respectively
<code>as</code>	Return as a set of GenomicRanges, or with each splice junction annotated as a GenomicInteraction
<code>...</code>	Not used

**Details**

A canonical splice junction consists of a donor site and an acceptor site at each end of an intron, with a branching site somewhere within the intron. Canonical donor sites are 8nt long with the first two bases being exonic and the next 6 being derived from intronic sequences. Canonical acceptor sites are 5nt long with the first four bases being intronic and the final base being the first base of the next exon.

This functions uses each set of exons within a transcript to identify both donor and acceptor sites. Branch sites are not identified.

**Value**

A GRanges object with requested columns, and an additional column, 'site', annotating each region as a donor or acceptor site.

Alternatively, by specifying `as = "GInteractions"`, the junctions can be returned with each splice junction annotated as a GenomicInteraction. This can make the set of junctions easier to interpret for a given transcript.

**Examples**

```
library(rtracklayer)
gtf_cols <- c(
  "transcript_id", "transcript_name", "gene_id", "gene_name", "exon_number"
)
gtf <- import.gff(
  system.file("extdata/gencode.v44.subset.gtf.gz", package = "transmogR"),
  feature.type = "exon", colnames = gtf_cols
)
sj <- sjFromExons(gtf)
sj

## Or to simplify shared splice junctions across multiple transcripts
library(extraChIPs, quietly = TRUE)
chopMC(sj)

## Splice Junctions can also be returned as a GInteractions object with
## anchorOne as the donor & anchorTwo as the acceptor sites
sjFromExons(gtf, as = "GInteractions")
```

transmogrify

*Mogrify a transcriptome using a set of variants***Description**

Use a set of SNPs, insertions and deletions to modify a reference transcriptome

**Usage**

```
transmogrify(x, var, exons, ...)
```

```
## S4 method for signature 'XStringSet,GRanges,GRanges'
```

```
transmogrify(
  x,
  var,
  exons,
  alt_col = "ALT",
  trans_col = "transcript_id",
  omit_ranges = NULL,
  tag = NULL,
  sep = "_",
  var_tags = FALSE,
  var_sep = "_",
  ol_vars = "fail",
  verbose = TRUE,
  mc.cores = 1,
  ...
)
```

```
## S4 method for signature 'BSgenome,GRanges,GRanges'
```

```
transmogrify(
  x,
  var,
  exons,
  alt_col = "ALT",
  trans_col = "transcript_id",
  omit_ranges = NULL,
  tag = NULL,
  sep = "_",
  var_tags = FALSE,
  var_sep = "_",
  ol_vars = "fail",
  verbose = TRUE,
  mc.cores = 1,
  ...
)
```

```
## S4 method for signature 'BSgenome,VcfFile,GRanges'
```

```
transmogrify(
  x,
  var,
```



```

    exons,
    alt_col = "ALT",
    trans_col = "transcript_id",
    omit_ranges = NULL,
    tag = NULL,
    sep = "_",
    var_tags = FALSE,
    var_sep = "_",
    ol_vars = "fail",
    verbose = TRUE,
    mc.cores = 1,
    which,
    ...
)

## S4 method for signature 'XStringSet,VcfFile,GRanges'
transmogrify(
  x,
  var,
  exons,
  alt_col = "ALT",
  trans_col = "transcript_id",
  omit_ranges = NULL,
  tag = NULL,
  sep = "_",
  var_tags = FALSE,
  var_sep = "_",
  ol_vars = "fail",
  verbose = TRUE,
  mc.cores = 1,
  which,
  ...
)

```

### Arguments

x	Reference genome as either a DNASTringSet or BSgenome
var	GRanges object containing the variants
exons	GRanges object with ranges representing exons
...	Passed to <a href="#">parallel::mclapply</a>
alt_col	Column from var containing alternate bases
trans_col	Column from 'exons' containing the transcript_id
omit_ranges	GRanges object containing ranges to omit, such as PAR-Y regions, for example
tag	Optional tag to add to all sequence names which were modified
sep	Separator to place between seqnames names & tag
var_tags	logical(1) Add tags indicating which type of variant were incorporated, with 's', 'i' and 'd' representing SNPs, Insertions and Deletions respectively
var_sep	Separator between any previous tags and variant tags
ol_vars	Error handling for any overlapping variants. See <a href="#">cleanVariants</a> for possible values and an explanation

verbose	logical(1) Include informative messages, or operate silently
mc.cores	Number of cores to be used when multi-threading via <a href="#">parallel::mclapply</a>
which	GRanges object passed to <a href="#">VariantAnnotation::ScanVcfParam</a> if using a VCF directly

## Details

Produce a set of variant modified transcript sequences from a standard reference genome. Supported variants are SNPs, Insertions and Deletions

Ranges needing to be masked, such as the Y-chromosome, or Y-PAR can be provided.

**It should be noted that this is a time consuming process** Inclusion of a large set of insertions and deletions across an entire transcriptome can involve individually modifying many thousands of transcripts, which can be a computationally demanding task. Whilst this can be parallelised using an appropriate number of cores, this may also prove taxing for lower power laptops, and pre-emptively closing memory hungry programs such as Slack, or internet browsers may be prudent.

## Value

An XStringSet

## Examples

```
library(GenomicRanges)
library(GenomicFeatures)
seq <- DNASTringSet(c(chr1 = "ACGTAAATGG"))
exons <- GRanges(c("chr1:1-3:-", "chr1:7-9:-"))
exons$transcript_id <- c("trans1")

# When using extractTranscriptSeqs -stranded exons need to be sorted by end
exons <- sort(exons, decreasing = TRUE, by = ~end)
exons
trByExon <- splitAsList(exons, exons$transcript_id)

# Check the sequences
seq
extractTranscriptSeqs(seq, trByExon)

# Define some variants
var <- GRanges(c("chr1:2", "chr1:8"))
var$ALT <- c("A", "GGG")

# Include the variants adding tags to indicate a SNP and indel
# The exons GRanges object will be split by transcript internally
transmoglify(seq, var, exons, var_tags = TRUE)
```

---

upsetVarByCol

---

*Show Variants by Impacted Columns*


---

## Description

Produce an UpSet plot showing unique values from a given column

**Usage**

```
upsetVarByCol(
  gr,
  var,
  alt_col = "ALT",
  mcol = "transcript_id",
  ...,
  fill = NULL,
  fill_scale = scale_fill_discrete(),
  expand_sets = 0.2,
  hj_sets = 1.1,
  expand_intersect = 0.1,
  vj_intersect = -0.5,
  label_size = 3.5,
  title
)
```

**Arguments**

<code>gr</code>	GRanges object with ranges representing a key feature such as exons
<code>var</code>	GRanges object with variants in a given column
<code>alt_col</code>	Column within <code>var</code> containing the alternate allele
<code>mcol</code>	The column within <code>gr</code> to summarise results by
<code>...</code>	Passed to <a href="#">SimpleUpset::simpleUpSet</a>
<code>fill</code>	Optional column in <code>gr</code> used to fill intersections and sets
<code>fill_scale</code>	Discrete ggplot2 scale for filling bars. Ignored if <code>fill = NULL</code>
<code>expand_sets</code>	Expand the set-size x-axis by this amount
<code>hj_sets</code>	Horizontal adjustment of set size labels
<code>expand_intersect</code>	Expand the intersection y-axis by this amount
<code>vj_intersect</code>	Vertical adjustment of intersection size labels
<code>label_size</code>	Control the size of both intersection and set size labels
<code>title</code>	Summary title to show above the intersection panel. Can be hidden by setting to <code>NULL</code>

**Details**

Take a set of variants, classify them as SNV, Insertion and Deletion, then using a GRanges object, produce an UpSet plot showing impacted values from a given column

**Value**

An UpSet plot

**See Also**

[SimpleUpset::simpleUpSet](#)

## Examples

```
library(rtracklayer)
library(VariantAnnotation)
library(ggplot2)
gtf <- import.gff(
  system.file("extdata/gencode.v44.subset.gtf.gz", package = "transmogR"),
  feature.type = "exon"
)
vcf <- system.file("extdata/1000GP_subset.vcf.gz", package = "transmogR")
var <- rowRanges(readVcf(vcf, param = ScanVcfParam(fixed = "ALT")))
upsetVarByCol(gtf, var)
upsetVarByCol(
  gtf, var, fill = "transcript_type",
  fill_scale = scale_fill_brewer(palette = "Set1")
)
```

---

varTags

---

Create a set of tags indicating overlap status with variants

---

## Description

Create a set of tags indicating overlap status with variants

## Usage

```
varTags(x, var, tag = NULL, var_tags = TRUE, sep = "_", pre = sep, ...)
```

## Arguments

x	GRanges or GRangesList
var	Set of variants for x to be compared to
tag	Tag to be added for all overlapping positions
var_tags	logical(1) Include 's', 'i' and 'd' tags. See details
sep	Separator added between tag and var_tags
pre	Separator to add at the start of returned tags
...	Passed to <a href="#">cleanVariants()</a>

## Details

Take a GRanges or GRangesList and compare against a set of variants. Variants will be classified into SNV, Insertions and Deletions using [varTypes\(\)](#) and tags defined. An overall set of tags defining any overlap can be created by themselves. An additional set of tags containing 's', 'i' or 'd' to indicate overlap with an SNV, Insertion or Deletion can also be created, with the concatenation of both tags being returned.

## Value

Character vector of the same length as x

**Examples**

```
# Load the included subset of 1000 Genomes Variants
library(VariantAnnotation)
vcf <- system.file("extdata/1000GP_subset.vcf.gz", package = "transmogR")
vcf <- VcfFile(vcf)
var <- cleanVariants(vcf)
# Now load some exons, then split by transcript, subsetting to the first 40
library(rtracklayer)
f <- system.file("extdata/gencode.v44.subset.gtf.gz", package = "transmogR")
gtf <- import.gff(f, feature.type = "exon")
exon_by_trans <- splitAsList(gtf, gtf$transcript_id)[1:40]
# And produce tags based on the overlapping variants within the exons
# Overlapping SNVs will return an 's' whilst insertions include an 'i'
varTags(exon_by_trans, var, tag = "1000GP")
```

varTypes

*Identify SNVs, Insertions and Deletions***Description**

Identify SNVs, Insertions and Deletions within a GRanges object

**Usage**

```
varTypes(x, alt_col = "ALT", ...)
```

**Arguments**

x	GenomicRanges object
alt_col	Name of the column with mcols(x) which contains the alternate allele. Can be an XStringSetList, XStringSet or character
...	Not used

**Details**

Using the width of the reference and alternate alleles, classify each range as an SNV, Insertion or Deletion.

- SNVs are expected to have REF & ALT widths of 1
- Insertions are expected to have ALT longer than REF
- Deletions are expected to have ALT shorter than REF

These are relatively permissive criteria

**Value**

Character vector

**Examples**

```
# Load the example VCF and classify ranges
library(VariantAnnotation)
f <- system.file("extdata/1000GP_subset.vcf.gz", package = "transmogR")
vcf <- readVcf(f)
gr <- rowRanges(vcf)
type <- varTypes(gr)
table(type)
gr[type != "SNV"]
```

# Index

## \* internal

transmogR-package, 2

Biostrings::replaceLetterAt(), 12

Biostrings::XString, 10

Biostrings::XStringSet, 10

cleanVariants, 3, 8, 9, 12, 17

cleanVariants(), 20

cleanVariants, GRanges-method  
(cleanVariants), 3

cleanVariants, VcfFile-method  
(cleanVariants), 3

cleanVariants-methods (cleanVariants), 3

digestSalmon, 4

edgeR::catchSalmon(), 5

genomogrify, 6

genomogrify(), 2, 10, 12, 14

genomogrify, BSgenome, GRanges-method  
(genomogrify), 6

genomogrify, BSgenome, VcfFile-method  
(genomogrify), 6

genomogrify, XStringSet, GRanges-method  
(genomogrify), 6

genomogrify, XStringSet, VcfFile-method  
(genomogrify), 6

genomogrify-methods (genomogrify), 6

indelcator, 9

indelcator(), 2

indelcator, BSgenome, GRanges-method  
(indelcator), 9

indelcator, DNASTringSet, GRanges-method  
(indelcator), 9

indelcator, XString, GRanges-method  
(indelcator), 9

match.arg(), 13

overdispFromBoots (digestSalmon), 4

overlapsByVar, 10

overlapsByVar(), 2

overlapsByVar, GRanges, GRanges-method  
(overlapsByVar), 10

overlapsByVar, GRangesList, GRanges-method  
(overlapsByVar), 10

overlapsByVar-methods (overlapsByVar),  
10

owl, 11

owl(), 2

owl, BSgenome, GRanges-method (owl), 11

owl, XStringSet, GRanges-method (owl), 11

parallel::mclapply, 8, 9, 17, 18

parallel::mclapply(), 14

parY, 12

parY(), 3

parY, character-method (parY), 12

parY, Seqinfo-method (parY), 12

parY-methods (parY), 12

rowSums, 11

shiftByVar, 13

SimpleUpset::simpleUpSet, 19

sjFromExons, 14

transmogR (transmogR-package), 2

transmogR-package, 2

transmogRify, 16

transmogRify(), 2, 10, 12

transmogRify, BSgenome, GRanges, GRanges-method  
(transmogRify), 16

transmogRify, BSgenome, VcfFile, GRanges-method  
(transmogRify), 16

transmogRify, XStringSet, GRanges, GRanges-method  
(transmogRify), 16

transmogRify, XStringSet, VcfFile, GRanges-method  
(transmogRify), 16

transmogRify-methods (transmogRify), 16

upsetVarByCol, 18

upsetVarByCol(), 3

VariantAnnotation::ScanVcfParam, 4, 8,  
18

VariantAnnotation::VcfFile, 3, 8

varTags, [20](#)

varTypes, [21](#)

varTypes(), [3](#), [20](#)