Package 'StructuralVariantAnnotation'

March 30, 2021

Type Package

Title Variant annotations for structural variants

Version 1.6.0

Date 2020-04-15

Description StructuralVariantAnnotation contains useful helper functions for dealing with structural variants in VCF format. The packages contains functions for parsing VCFs from a number of popular callers as well as functions for dealing with breakpoints involving two separate genomic loci encoded as GRanges objects.

License GPL-3

Depends GenomicRanges, rtracklayer, VariantAnnotation, BiocGenerics, R (>= 3.6.0)

Imports assertthat, Biostrings, stringr, dplyr, methods, rlang

Suggests BSgenome.Hsapiens.UCSC.hg19, ggplot2, devtools, testthat, roxygen2, covr, knitr, plyranges, ggbio, biovizBase, circlize, tictoc, GenomeInfoDb, IRanges, S4Vectors, SummarizedExperiment

RoxygenNote 7.1.0

Encoding UTF-8

VignetteBuilder knitr

biocViews DataImport, Sequencing, Annotation, Genetics, VariantAnnotation

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

git_branch RELEASE_3_12

git_last_commit e585636

git_last_commit_date 2020-10-27

Date/Publication 2021-03-29

Author Daniel Cameron [aut, cre] (<https://orcid.org/0000-0002-0951-7116>), Ruining Dong [aut] (<https://orcid.org/0000-0003-1433-0484>)

Maintainer Daniel Cameron <daniel.l.cameron@gmail.com>

R topics documented:

`.constrict` *constrict*

Description

constrict

Usage

```
.constrict(gr, ref = NULL, position = "middle")
```

Arguments

`gr`	GRanges object
`ref`	reference
`position`	only 'middle' position is accepted.

Value

A constricted GRanges object.

`.isSymbolic` *Determining whether the variant is a symbolic allele.*

Description

Determining whether the variant is a symbolic allele.

Usage

```
.isSymbolic(r, a)
```

Arguments

`r`	Reference vector.
`a`	ALT vector.

Value

A logical list of which the length is the same with the input object.

`.pairwiseLCPrefix` *vectorised pairwise longest common prefix Returns the length of the longest common prefix for each string pair*

Description

vectorised pairwise longest common prefix Returns the length of the longest common prefix for each string pair

Usage

```
.pairwiseLCPrefix(s1, s2, ignore.case = FALSE)
```

Arguments

`s1, s2`	A pair of strings.
`ignore.case`	Whether cases in the strings should be ignored.

Value

The length of the longest common prefix for each string pair.

.svLen — *Returns the structural variant length of the first allele*

Description

Returns the structural variant length of the first allele

Usage

```
.svLen(vcf)
```

Arguments

vcf	VCF object

Value

Structural variant lengths of the first allele.

.testfile — *Testthat helper utility to locate files used for package tests*

Description

Testthat helper utility to locate files used for package tests

Usage

```
.testfile(filename, location = "extdata")
```

Arguments

filename	Name of the test file.
location	Directory of the test file.

Value

Returns the file to be tested.

.testrecord *Loading a VCF containing the given records*

Description

Loading a VCF containing the given records

Usage

```
.testrecord(record)
```

Arguments

record	string vector of record to write

Value

A VCF object.

.toVcfBreakendNotationAlt *Converts to breakend notation*

Description

Converts to breakend notation

Usage

```
.toVcfBreakendNotationAlt(gr, insSeq = gr$insSeq, ref = gr$REF)
```

Arguments

gr	GRanges object.
insSeq	insert sequence of the GRanges.
ref	reference sequence of the GRanges.

Value

breakendAlt or breakpointAlt depending on whether the variant is partnered.

.unXStringSet *converts an XStringSet to a character*

Description

converts an XStringSet to a character

Usage

```
.unXStringSet(x)
```

Arguments

x	An XStringSet.

Value

A character.

align_breakpoints *Adjusting the nominal position of a pair of partnered breakpoint.*

Description

Adjusting the nominal position of a pair of partnered breakpoint.

Usage

```
align_breakpoints(
  vcf,
  align = c("centre"),
  is_higher_breakend = names(vcf) < info(vcf)$PARID
)
```

Arguments

vcf	A VCF object.
align	The alignment type.
is_higher_breakend	Breakpoint ID ordering.

Value

A VCF object with adjusted nominal positions.

`breakendRanges` *Extracting unpartnered breakend structural variants as a GRanges*

Description

Extracting unpartnered breakend structural variants as a GRanges

Usage

```
breakendRanges(x, ...)

## S4 method for signature 'VCF'
breakendRanges(x, ...)
```

Arguments

`x`	A VCF object.
`...`	Parameters of `.breakpointRanges()`. See breakpointRanges for more details.

Details

The VCF standard supports single breakends where a breakend is not part of a novel adjacency and lacks a mate. This function supports parsing single breakends to GRanges, where a dot symbol is used in the ALT field to annotate the directional information. Single breakends provide insights to situations when one side of the structural variant is not observed, due to e.g. low mappability, non-reference contigs, complex multi-break operations, etc. See Section 5.4.9 of https://samtools.github.io/hts-specs/VCFv4.3.pdf for details of single breakends.

Value

A GRanges object of SVs.

Methods (by class)

- `VCF`: Extracting unpartnered structural variants as GRanges.

Examples

```
vcf.file <- system.file("extdata", "gridss.vcf",
                        package = "StructuralVariantAnnotation")
vcf <- VariantAnnotation::readVcf(vcf.file, "hg19")
breakendRanges(vcf)
breakendRanges(vcf, nominalPosition=TRUE)
```

breakpointgr2bedpe *Converting breakpoint GRanges to BEDPE-like dataframe*

Description

Converting breakpoint GRanges to BEDPE-like dataframe

Usage

```
breakpointgr2bedpe(gr)
```

Arguments

gr	A GRanges object.

Details

breakpointgr2bedpe converts a breakpoint GRanges to a BEDPE-formatted dataframe. The BEDPE format consists of two sets of genomic loci, optinal columns of name, score, strand1, strand2 and any user-defined fields. See https://bedtools.readthedocs.io/en/latest/content/general-usage.html for more details of BEDPE format.

Value

A BEDPE-formatted data frame.

Examples

```
#coverting a GRanges object to BEDPE-like dataframe
vcf.file <- system.file("extdata", "gridss.vcf", package = "StructuralVariantAnnotation")
vcf <- VariantAnnotation::readVcf(vcf.file, "hg19")
gr <- breakpointRanges(vcf)
breakpointgr2bedpe(gr)
```

breakpointgr2pairs *Converts a breakpoint GRanges object to a Pairs object*

Description

Converts a breakpoint GRanges object to a Pairs object

Converts a BEDPE Pairs containing pairs of GRanges loaded using to a breakpoint GRanges object.

Usage

```
breakpointgr2pairs(
  bpgr,
  writeQualAsScore = TRUE,
  writeName = TRUE,
  bedpeName = NULL,
  firstInPair = NULL
)

pairs2breakpointgr(
  pairs,
  placeholderName = "bedpe",
  firstSuffix = "_1",
  secondSuffix = "_2",
  nameField = "name",
  renameScoreToQUAL = TRUE
)
```

Arguments

bpgr	breakpoint GRanges object
writeQualAsScore	write the breakpoint GRanges QUAL field as the score fields for compatibility with BEDPE rtracklayer export
writeName	write the breakpoint GRanges QUAL field as the score fields for compatibility with BEDPE rtracklayer export
bedpeName	function that returns the name to use for the breakpoint. Defaults to the sourceId, name column, or row names (in that priority) of the first breakend of each pair.
firstInPair	function that returns TRUE for breakends that are considered the first in the pair, and FALSE for the second in pair breakend. By default, the first in the pair is the breakend with the lower ordinal in the breakpoint GRanges object.
pairs	a Pairs object consisting of two parallel genomic loci.
placeholderName	prefix to use to ensure each entry has a unique ID.
firstSuffix	first in pair name suffix to ensure breakend name uniqueness
secondSuffix	second in pair name suffix to ensure breakend name uniqueness
nameField	Fallback field for row names if the Pairs object does not contain any names. BEDPE files loaded using rtracklayer use the "name" field.
renameScoreToQUAL	renames the 'score' column to 'QUAL'. Performing this rename results in a consistent variant quality score column name for variant loaded from BEDPE and VCF.

Details

Breakpoint-level column names will override breakend-level column names.

Value

Pairs GRanges object suitable for export to BEDPE by rtracklayer

Breakpoint GRanges object.

Examples

```
vcf.file <- system.file("extdata", "gridss.vcf", package = "StructuralVariantAnnotation")
bpgr <- breakpointRanges(VariantAnnotation::readVcf(vcf.file))
pairgr <- breakpointgr2pairs(bpgr)
rtracklayer::export(pairgr, con="example.bedpe")
bedpe.file <- system.file("extdata", "gridss.bedpe", package = "StructuralVariantAnnotation")
bedpe.pairs <- rtracklayer::import(bedpe.file)
bedpe.bpgr <- pairs2breakpointgr(bedpe.pairs)
```

`breakpointGRangesToVCF` *Converts the given breakpoint GRanges object to VCF format in breakend notation.*

Description

Converts the given breakpoint GRanges object to VCF format in breakend notation.

Usage

```
breakpointGRangesToVCF(gr, ...)
```

Arguments

`gr`	breakpoint GRanges object. Can contain both breakpoint and single breakend SV records.
`...`	For cbind and rbind a list of VCF objects. For all other methods ... are additional arguments passed to methods. See VCF class in VariantAnnotation for more details.

Value

A VCF object.

`breakpointRanges` *Extracting the structural variants as a GRanges.*

Description

Extracting the structural variants as a GRanges.

.breakpointRanges() is an internal function for extracting structural variants as GRanges.

Usage

```
breakpointRanges(x, ...)

## S4 method for signature 'VCF'
breakpointRanges(x, ...)

.breakpointRanges(
  vcf,
  nominalPosition = FALSE,
  placeholderName = "svrecord",
  suffix = "_bp",
  info_columns = NULL,
  unpartneredBreakends = FALSE,
  inferMissingBreakends = FALSE
)
```

Arguments

x	A VCF object
...	Parameters of `.breakpointRanges()`. See below.
vcf	A VCF object.
nominalPosition	Determines whether to call the variant at the nominal VCF position, or to call the confidence interval (incorporating any homology present). Default value is set to FALSE, where the interval is called based on the CIPOS tag. When set to TRUE, the ranges field contains the nomimal variant position only.
placeholderName	Variant name prefix to assign to unnamed variants.
suffix	The suffix to append to varaint names.
info_columns	VCF INFO columns to include in the GRanges object.
unpartneredBreakends	Determining whether to report unpartnered breakends. Default is set to FALSE.
inferMissingBreakends	Infer missing breakend records from ALT field of records without matching partners

Details

Structural variants are converted to breakend notation. Due to ambiguities in the VCF specifications, structural variants with multiple alt alleles are not supported. The CIPOS tag describes the uncertainty interval of the around the postition of the breakend. See Section 5.4.8 of https://samtools.github.io/hts-specs/VCFv4.3.pdf for details of CIPOS. If HOMLEN or HOMSEQ is defined without CIPOS, it is assumed that the variant position is left aligned. A breakend on the '+' strand indicates a break immediately after the given position, to the left of which is the DNA segment involved in the breakpoint. The '-' strand indicates a break immediately before the given position, rightwards of which is the DNA segment involved in the breakpoint. Unpaired variants are removed at this stage.

Value

A GRanges object of SVs.

Methods (by class)

- VCF: Extracting structural variants as GRanges.

Examples

```
vcf.file <- system.file("extdata", "vcf4.2.example.sv.vcf",
                        package = "StructuralVariantAnnotation")
vcf <- VariantAnnotation::readVcf(vcf.file, "hg19")
breakpointRanges(vcf)
breakpointRanges(vcf, nominalPosition=TRUE)
```

---

calculateReferenceHomology

Calculates the length of inexact homology between the breakpoint sequence and the reference

---

Description

Calculates the length of inexact homology between the breakpoint sequence and the reference

Usage

```
calculateReferenceHomology(
  gr,
  ref,
  anchorLength = 300,
  margin = 5,
  match = 2,
  mismatch = -6,
  gapOpening = 5,
  gapExtension = 3
)
```

Arguments

gr	reakpoint GRanges
ref	reference BSgenome
anchorLength	Number of bases to consider for homology
margin	Number of additional reference bases include. This allows for inexact homology to be detected even in the presence of indels.
match	see Biostrings::pairwiseAlignment
mismatch	see Biostrings::pairwiseAlignment
gapOpening	see Biostrings::pairwiseAlignment
gapExtension	see Biostrings::pairwiseAlignment

Value

A dataframe containing the length of inexact homology between the breakpoint sequence and the reference.

`countBreakpointOverlaps` *Counting overlapping breakpoints between two breakpoint sets*

Description

Counting overlapping breakpoints between two breakpoint sets

Usage

```
countBreakpointOverlaps(
  querygr,
  subjectgr,
  countOnlyBest = FALSE,
  breakpointScoreColumn = "QUAL",
  maxgap = -1L,
  minoverlap = 0L,
  ignore.strand = FALSE,
  sizemargin = NULL,
  restrictMarginToSizeMultiple = NULL
)
```

Arguments

`querygr`, `subjectgr`, `maxgap`, `minoverlap`, `ignore.strand`, `sizemargin`, `restrictMarginToSizeMultiple`
: See `findBreakpointOverlaps()`.

`countOnlyBest`
: Default value set to FALSE. When set to TRUE, the result count each subject breakpoint as overlaping only the best overlapping query breakpoint. The best breakpoint is considered to be the one with the highest QUAL score.

`breakpointScoreColumn`
: Query column defining a score for determining which query breakpoint is considered the best when countOnlyBest=TRUE.

Details

`countBreakpointOverlaps()` returns the number of overlaps between breakpoint objects, based on the output of `findBreakpointOverlaps()`. See GenomicRanges::countOverlaps-methods

Value

An integer vector containing the tabulated query overlap hits.

Examples

```
truth_vcf = VariantAnnotation::readVcf(system.file("extdata", "na12878_chr22_Sudmunt2015.vcf", package = "Str
crest_vcf = VariantAnnotation::readVcf(system.file("extdata", "na12878_chr22_crest.vcf", package = "Structura
caller_bpgr = breakpointRanges(crest_vcf)
caller_bpgr$true_positive = countBreakpointOverlaps(caller_bpgr, breakpointRanges(truth_vcf),
  maxgap=100, sizemargin=0.25, restrictMarginToSizeMultiple=0.5, countOnlyBest=TRUE)
```

`elementExtract` *Extracts the element of each element at the given position*

Description

Extracts the element of each element at the given position

Usage

```
elementExtract(x, offset = 1)
```

Arguments

`x`	list-like object
`offset`	offset of list

Value

The element of each element at given positions.

`extractBreakpointSequence` *Extracts the breakpoint sequence.*

Description

Extracts the breakpoint sequence.

Usage

```
extractBreakpointSequence(gr, ref, anchoredBases, remoteBases = anchoredBases)
```

Arguments

`gr`	breakpoint GRanges
`ref`	Reference BSgenome
`anchoredBases`	Number of bases leading into breakpoint to extract
`remoteBases`	Number of bases from other side of breakpoint to extract

Details

The sequence is the sequenced traversed from the reference anchor bases to the breakpoint. For backward (-) breakpoints, this corresponds to the reverse compliment of the reference sequence bases.

Value

Breakpoint sequence around the variant position.

`extractReferenceSequence` *Returns the reference sequence around the breakpoint position*

Description

Returns the reference sequence around the breakpoint position

Usage

```
extractReferenceSequence(
  gr,
  ref,
  anchoredBases,
  followingBases = anchoredBases
)
```

Arguments

`gr`	breakpoint GRanges
`ref`	Reference BSgenome
`anchoredBases`	Number of bases leading into breakpoint to extract
`followingBases`	Number of reference bases past breakpoint to extract

Details

The sequence is the sequenced traversed from the reference anchor bases to the breakpoint. For backward (-) breakpoints, this corresponds to the reverse compliment of the reference sequence bases.

Value

Reference sequence around the breakpoint position.

`findBreakpointOverlaps` *Finding overlapping breakpoints between two breakpoint sets*

Description

Finding overlapping breakpoints between two breakpoint sets

Usage

```
findBreakpointOverlaps(
  query,
  subject,
  maxgap = -1L,
  minoverlap = 0L,
  ignore.strand = FALSE,
  sizemargin = NULL,
  restrictMarginToSizeMultiple = NULL
)
```

Arguments

query, subject	Both of the input objects should be GRanges objects. Unlike findOverlaps(), subject cannot be ommitted. Each breakpoint must be accompanied with a partner breakend, which is also in the GRanges, with the partner's id recorded in the partner field. See GenomicRanges::findOverlaps-methods for details.
maxgap, minoverlap	Valid overlapping thresholds of a maximum gap and a minimum overlapping positions between breakend intervals. Both should be scalar integers. maxgap allows non-negative values, and minoverlap allows positive values. See GenomicRanges::findOverlaps-methods for details.
ignore.strand	Default value is FALSE. strand information is ignored when set to TRUE. See GenomicRanges::findOverlaps-methods for details.
sizemargin	Error margin in allowable size to prevent matching of events of different sizes, e.g. a 200bp event matching a 1bp event when maxgap is set to 200.
restrictMarginToSizeMultiple	Size restriction multiplier on event size. The default value of 0.5 requires that the breakpoint positions can be off by at maximum, half the event size. This ensures that small deletion do actually overlap at least one base pair.

Details

findBreakpointOverlaps() is an efficient adaptation of findOverlaps-methods() for breakend ranges. It searches for overlaps between breakpoint objects, and return a matrix including index of overlapping ranges as well as error stats. All breakends must have their partner breakend included in the partner field. A valid overlap requires that breakends on boths sides meets the overlapping requirements.

See GenomicRanges::findOverlaps-methods for details of overlap calculation.

Value

A dataframe containing index and error stats of overlapping breakpoints.

Examples

```
#reading in VCF files
query.file <- system.file("extdata", "gridss-na12878.vcf", package = "StructuralVariantAnnotation")
subject.file <- system.file("extdata", "gridss.vcf", package = "StructuralVariantAnnotation")
query.vcf <- VariantAnnotation::readVcf(query.file, "hg19")
subject.vcf <- VariantAnnotation::readVcf(subject.file, "hg19")
#parsing vcfs to GRanges objects
```

```
query.gr <- breakpointRanges(query.vcf)
subject.gr <- breakpointRanges(subject.vcf)
#find overlapping breakpoint intervals
findBreakpointOverlaps(query.gr, subject.gr)
findBreakpointOverlaps(query.gr, subject.gr, ignore.strand=TRUE)
findBreakpointOverlaps(query.gr, subject.gr, maxgap=100, sizemargin=0.5)
```

`findInsDupOverlaps` *Finds duplication events that are reported as inserts. As sequence alignment algorithms do no allow backtracking, long read-based variant callers will frequently report small duplication as insertion events. Whilst both the duplication and insertion representations result in the same sequence, this representational difference is problematic when comparing variant call sets.*

Description

WARNING: this method does not yet check that the inserted sequence actually matched the duplicated sequence.

Usage

```
findInsDupOverlaps(query, subject, maxgap = -1L, maxsizedifference = 0L)
```

Arguments

`query`	a breakpoint GRanges object
`subject`	a breakpoint GRanges object
`maxgap`	maximum distance between the insertion position and the duplication
`maxsizedifference`	maximum size difference between the duplication and insertion.

Value

Hits object containing the ordinals of the matching breakends in the query and subject

---

`isStructural` *Determining whether the variant is a structural variant*

Description

Determining whether the variant is a structural variant

Usage

```
isStructural(x, ...)

## S4 method for signature 'CollapsedVCF'
isStructural(x, ..., singleAltOnly = TRUE)

## S4 method for signature 'ExpandedVCF'
isStructural(x, ...)

## S4 method for signature 'VCF'
isStructural(x, ...)
```

Arguments

x	A VCF object.
...	Internal parameters.
singleAltOnly	Whether only single ALT values are accepted. Default is set to TRUE.

Details

The function takes a VCF object as input, and returns a logical value for each row, determining whether the variant is a structural variant.

Value

A logical list of which the length is the same with the input object.

Methods (by class)

- CollapsedVCF: Determining whether a CollapsedVCF object is a strucrual variant. Only single ALT values are accepted.
- ExpandedVCF: Determining whether a ExpandedVCF object is a structural variant.
- VCF: Determining whether a VCF object is a structural variant.

Examples

```
vcf.file <- system.file("extdata", "gridss.vcf", package = "StructuralVariantAnnotation")
vcf <- VariantAnnotation::readVcf(vcf.file, "hg19")
isStructural(vcf)
```

---

isSymbolic *Determining whether the variant is a symbolic allele.*

Description

Determining whether the variant is a symbolic allele.

Usage

```
isSymbolic(x, ...)

## S4 method for signature 'CollapsedVCF'
isSymbolic(x, ..., singleAltOnly = TRUE)

## S4 method for signature 'ExpandedVCF'
isSymbolic(x, ...)
```

Arguments

x	A VCF object.
...	Internal parameters.
singleAltOnly	Whether only single ALT values are accepted. Default is set to TRUE.

Details

The function takes a VCF object as input, and returns a logical value for each row, determining whether the variant is a symbolic allele.

Value

A logical list of which the length is the same with the input object.

Methods (by class)

- CollapsedVCF: Determining whether a CollapsedVCF object is a symbolic allele. Only single ALT values are accepted.
- ExpandedVCF: Determining whether a ExpandedVCF object is a symbolic allele

Examples

```
vcf.file <- system.file("extdata", "gridss.vcf", package = "StructuralVariantAnnotation")
vcf <- VariantAnnotation::readVcf(vcf.file, "hg19")
isSymbolic(vcf)
```

---

partner *GRanges representing the breakend coordinates of structural variants #@export Partner breakend for each breakend.*

Description

GRanges representing the breakend coordinates of structural variants #@export Partner breakend for each breakend.

Usage

```
partner(gr, selfPartnerSingleBreakends = FALSE)
```

Arguments

gr	GRanges object of SV breakends
selfPartnerSingleBreakends	treat single breakends as their own partner.

Details

All breakends must have their partner breakend included in the GRanges.

Value

A GRanges object in which each entry is the partner breakend of those in the input object.

Examples

```
#reading in a VCF file as \code{vcf}
vcf.file <- system.file("extdata", "gridss.vcf", package = "StructuralVariantAnnotation")
vcf <- VariantAnnotation::readVcf(vcf.file, "hg19")
#parsing \code{vcf} to GRanges object \code{gr}
gr <- breakpointRanges(vcf)
#output partner breakend of each breakend in \code{gr}
partner(gr)
```

---

simpleEventLength *Length of event if interpreted as an isolated breakpoint.*

Description

Length of event if interpreted as an isolated breakpoint.

Usage

```
simpleEventLength(gr)
```

Arguments

gr	breakpoint GRanges object

Value

Length of the simplest explaination of this breakpoint/breakend.

`simpleEventType` *Type of simplest explaination of event. Possible types are: | Type | Description | | BND | Single breakend | | CTX | Interchromosomal translocation | | INV | Inversion. Note that both ++ and – breakpoint will be classified as inversion regardless of whether the matching breakpoint actually exists | | DUP | Tandem duplication | | INS | Insertion | | DEL | Deletion |*

Description

Type of simplest explaination of event. Possible types are: | Type | Description | | BND | Single breakend | | CTX | Interchromosomal translocation | | INV | Inversion. Note that both ++ and – breakpoint will be classified as inversion regardless of whether the matching breakpoint actually exists | | DUP | Tandem duplication | | INS | Insertion | | DEL | Deletion |

Usage

```
simpleEventType(gr, insertionLengthThreshold = 0.5)
```

Arguments

`gr`	breakpoint GRanges object
`insertionLengthThreshold`	portion of inserted bases compared to total event size to be classified as an insertion. For example, a 5bp deletion with 5 inserted bases will be classified as an INS event.

Value

Type of simplest explaination of event

`StructuralVariantAnnotation` *StructuralVariantAnnotation: a package for SV annotation*

Description

StructuralVariantAnnotation contains useful helper functions for reading and interpreting structural variants calls. The packages contains functions for parsing VCFs from a number of popular caller as well as functions for dealing with breakpoints involving two separate genomic loci. The package takes a 'GRanges' based breakend-centric approach.

Details

* Parse VCF objects with the 'breakpointRanges()' and 'breakendRanges()'functions. * Find breakpoint overlaps with the 'findBreakpointOverlaps()' and 'countBreakpointOverlaps()' functions. * Generate BEDPE files for circos plot with 'breakpointgr2pairs()' function. * ...

For more details on the features of StructuralVariantAnnotation, read the vignette: 'browseVignettes(package = "StructuralVariantAnnotation")'

%na% *Replaces the NA values in a with corresponding values in b*

Description

Replaces the NA values in a with corresponding values in b

Usage

```
a %na% b
```

Arguments

a, b	objects to be tested or coerced.

Value

The altered object.

%null% *Uses b if a is NULL*

Description

Uses b if a is NULL

Usage

```
a %null% b
```

Arguments

a, b	objects to be tested or coerced.

Value

An un-null object.

Index