R453Plus1Toolbox

October 25, 2011

`AVASet-class` *Class to contain Amplicon Variant Analyzer Output*

Description

Container to store data imported from a project of Roche's Amplicon Variant Analyzer Software. It stores all information into an extended version of the Biobase ExpressionSet.

Objects from the Class

Objects can be created by calls of the form `AVASet(filename)`. While filename is the path of the project data (i.e. a directory that contains the files and subdirectories "Amplicons/ProjectDef/ampliconsProject.txt", "Amplicons/Results/Variants/currentVariantDefs.txt", "Amplicons/Results/Variants", "Amplicons/Results/Align").

Slots

`assayData`**:** Object of class `AssayData`. Contains the number of reads and the total read depth for every variant and each sample in forward and reverse direction. Its column number equals `nrow(phenoData)`.

`featureData`**:** Object of class `AnnotatedDataFrame`. Contains information about the type, position and reference of each variant.

`phenoData`**:** Object of class `AnnotatedDataFrame`. Contains the sample-IDs and name, annotation and group of the read data for all samples. If available, the lane, pico titer plate(s) (PTP) or MID(s) of each sample are shown as well.

`assayDataAmp`**:** Object of class `AssayData`. Contains the number of reads for every amplicon and each sample in forward/reverse direction. Its column number equals `nrow(featureDataAmp)`.

`featureDataAmp`**:** Object of class `AnnotatedDataFrame`. Contains the primer sequences, reference sequences and the coordinates of the target regions for every amplicon.

`referenceSequences`**:** Object of class `AlignedRead`. If additional alignment information were computed via `alignShortReads`, this slot knows about the chromosome, position and the strand of each reference sequence.

`variantFilterPerc`**:** Object of class `numeric`. Contains a threshold to display only those variants, whose coverage (in percent) in forward and reverse direction in at least one sample is higher than this filter value. See `setVariantFilter` for details about setting this value.

`variantFilter`**:** Object of class `character`. Contains a vector of variant names whose coverage (in percent) in forward and reverse direction in at least one sample is higher than the filter value in `variantFilterPerc`.

`dirs`: Object of class `character`. Based on a directory given at instantiation of the object, it contains a vector of several directories containing all relevant AVA-project files.

`experimentData`: Object of class `MIAME`. Contains details of the experiment.

`annotation`: Object of class `character`. Label associated with the annotation package used in the experiment.

`protocolData`: Object of class `annotatedDataFrame`. Contains additional information about the samples.

`.__classVersion__`: Object of class `Versions`. Remembers the R and R453Toolbox version numbers used to created the AVASet instance.

Extends

Class `eSet`, directly. Class `VersionedBiobase`, by class "eSet", distance 2. Class `Versioned`, by class "eSet", distance 3.

Methods

object[i,j :] Allows subsetting an AVASet object by features (i) and samples (j).

assayDataAmp(object), assayDataAmp(object)<-value: Similar to `assayData` of the Biobase ExpressionSet, this function returns/replaces the amplicon assay data.

fDataAmp(object): Similar to `fData` of the Biobase ExpressionSet, this function returns the amplicon feature data as a data frame.

featureDataAmp(object), featureDataAmp(object)<-value: Similar to `featureData` of the Biobase ExpressionSet, this function returns/replaces the amplicon feature data and feature meta.

referenceSequences(object), referenceSequences(object)<-value: Returns/replaces the reference sequence slot.

alignShortReads(object, bsGenome): Retrieve the chromosomal positions of the amplicon sequences.

setVariantFilter(object): Sets the filter to display only those variants, whose coverage (in percent) in forward and reverse direction in at least one sample is higher than the given value.

getVariantPercentages(object) Computes the coverage for every variant over all reads (forward and/or reverse) and for each sample.

annotateVariants(object): Annotates given genomic variants. See `annotateVariants` for details.

htmlReport(object): Exports all (filtered) variant data into a html report. See `htmlReport` for details

Author(s)

Christoph Bartenhagen

See Also

`MapperSet-class`, `annotateVariants`, `alignShortReads`, `htmlReport`, `setVariantFilter`, `getVariantPercentages`

Examples

```
# sum up class structure
showClass("AVASet")

# load an AVA dataset containing 6 samples, 4 amplicons and 259 variants
data(avaSetExample)
avaSetExample

# show contents of assay, feature and pheno data
head(assayData(avaSetExample)$variantForwCount)
head(assayData(avaSetExample)$totalForwCount)
head(assayData(avaSetExample)$variantRevCount)
head(assayData(avaSetExample)$totalRevCount)
head(fData(avaSetExample))
pData(avaSetExample)
assayDataAmp(avaSetExample)
fDataAmp(avaSetExample)
referenceSequences(avaSetExample)
```

AVASet *Creating an AVASet*

Description

This function imports a project of Roche's Amplicon Variant Analyzer Software. It stores all information into an extended version of the Biobase eSet.

Usage

```
AVASet(dirname)
```

Arguments

dirname	The path of the project data (i.e. a directory that contains the files and subdirectories "Amplicons/ProjectDef/ampliconsProject.txt", "Amplicons/Results/Variants/currentVariantDef "Amplicons/Results/Variants", "Amplicons/Results/Align").

Details

An AVASet object consists of three slots to store data about
1. variants

variantForwCount/variantRevCount: Data frames that contain the number of reads with the respective variant in forward/reverse direction.

totalForwCount/totalRevCount: Data frames that contain the total coverage for every variant location in forward/reverse direction.

referenceSeq: Gives the identifier of the reference sequence.

variantBase/referenceBases: The bases changed in each variant.

start/end: The position of the variant on the reference sequence.

canonicalPattern/name: Short identifiers of a variant including the position and the bases changed.

2. amplicons

forwCount/revCount: Data frames that contain the number of reads for every amplicon and each sample in forward/reverse direction.

primer1,primer2: The primer sequences for every amplicon.

referenceSeqID: The identifier of the reference sequence.

targetStart/targetEnd: The coordinates of the target region.

3. reference sequences

If additional information has been loaded from Ensembl via `alignShortReads`, this slot knows about the chro

The structure of the variant and amplicon data is derived from the Biobase eSet and thus separated into assayData, phenoData and featureData. All information about the reference sequences is stored into an object of class `AlignedRead`.

The phenoData of the variants lists the sample-IDs and name, annotation and group of the read data for all samples. If available, the pico titer plate(s) (PTP) or MID(s) of each sample are shown as well.

Value

An instance of the AVASet class.

Author(s)

Christoph Bartenhagen

See Also

`MapperSet-class`, `alignShortReads`

Examples

```
# load an AVA dataset containing 6 samples, 4 amplicons and 259 variants
data(avaSetExample)
avaSetExample
```

AnnotatedVariants-class
Class "AnnotatedVariants"

Description

A class for storing annotation about variants. An object of this class is returned by the method `annotateVariants`. The class has not been designed to be created by users directly.

Details

The list encapsulated by this class has one element for each variant. Each element is a nested list with the elements `genes`, `transcripts`, `exons` and `snps`. All these elements are data frames listing genes, transcripts, exons or snps respectively that were affected by the variant. Use the example below to explore the data frames' contents.

Objects from the Class

Objects can be created by calls of the form `new("AnnotatedVariants")`. The method `annotateVariants` returns AnnotatedVariants-objects.

Slots

`annotatedVariants`**:** Object of class `"list"` with one entry for each variant.

Methods

annotatedVariants `signature(object = "AnnotatedVariants")`: Get the list with variants.

annotatedVariants<- `signature(object = "AnnotatedVariants", value = "list")`: Set a new list with variants.

names `signature(x = "AnnotatedVariants")`: Get the names of the with variants.

names<- `signature(x = "AnnotatedVariants", value = "character")`: Set the names of the variants.

Author(s)

Hans-Ulrich Klein

See Also

`annotateVariants`, `htmlReport`

Examples

```
variants = data.frame(
    start=c(106157528, 106154991,106156184),
    end=c(106157528, 106154994,106156185),
    chromosome=c("4", "4", "4"),
    strand=c("+", "+", "+"),
    seqRef=c("A", "ATAG", "---"),
    seqMut=c("G", "----", "ATA"),
```

```
    seqSur=c("TACAGAA", "TTTATAGATA", "AGC---TCC"),
    stringsAsFactors=FALSE)
rownames(variants) = c("snp", "del", "ins")
## Not run: av = annotateVariants(variants)
## Not run: annotatedVariants(av)[["snp"]]
```

`Breakpoints-class` *Class "Breakpoints"*

Description

Container to store chimeric reads that were clustered to putative breakpoints indicating structural variants. Related information like breakpoint position or alignment information about the chimeric reads is stored as well.

Objects from the Class

Objects can be created by calls of the form `new("Breakpoints", ...)`. Usually, objects will be created by calling the `detectBreakpoints` method. It is not intended that users create objects of this class manually.

All slots of this class can be found twice. One slot name ends with "C1" and the other "C2". The slots labeled with "C2" are empty until `mergeBreakpoints` has been called and contain information about putativly associated breakpoints detected by `mergeBreakpoints`.

Slots

`seqsC1`**:** Object of class `"list"` with one data frame for each breakpoint. The data frame stores all chimeric reads covering the first breakpoint together with the alignment information.

`seqsC2`**:** Object of class `"list"` with one data frame for each breakpoint. The data frame stores all chimeric reads covering the second breakpoint together with the alignment information.

`commonBpsC1`**:** Object of class `"list"` with one data frame for each breakpoint. The data frame stores the consensus breakpoint sequence as well as the breakpoint coordinates of the first breakpoint.

`commonBpsC2`**:** Object of class `"list"` with one data frame for each breakpoint. The data frame stores the consensus breakpoint sequence as well as the breakpoint coordinates of the second breakpoint.

`commonAlignC1`**:** Object of class `"list"` with one object of class `PairwiseAlignedFixedSubject-class` for each breakpoint storing the alignments of the chimeric reads against the consensus breakpoint sequence for the first breakpoint.

`commonAlignC2`**:** Object of class `"list"` with one object of class `PairwiseAlignedFixedSubject-class` for each breakpoint storing the alignments of the chimeric reads against the consensus breakpoint sequence for the second breakpoint.

`alignedReadsC1`**:** Object of class `"list"` with one object of class `AlignedRead-class` storing all chimeric reads covering the first breakpoint and their alignments.

`alignedReadsC2`**:** Object of class `"list"` with one object of class `AlignedRead-class` storing all chimeric reads covering the second breakpoint and their alignments.

Methods

alignedReadsC1<- `signature(object = "Breakpoints", value = "list")`: Setter-method for the alignedReadsC1 slot.

alignedReadsC1 `signature(object = "Breakpoints")`: Getter-method for the alignedReadsC1 slot.

alignedReadsC2<- `signature(object = "Breakpoints", value = "list")`: Setter-method for the alignedReadsC2 slot.

alignedReadsC2 `signature(object = "Breakpoints")`: Getter-method for the alignedReadsC2 slot.

commonAlignC1<- `signature(object = "Breakpoints", value = "list")`: Setter-method for the commonAlignC1 slot.

commonAlignC1 `signature(object = "Breakpoints")`: Getter-method for the commonAlignC1 slot.

commonAlignC2<- `signature(object = "Breakpoints", value = "list")`: Setter-method for the commonAlignC2 slot.

commonAlignC2 `signature(object = "Breakpoints")`: Getter-method for the commonAlignC2 slot.

commonBpsC1<- `signature(object = "Breakpoints", value = "list")`: Setter-method for the commonBpsC1 slot.

commonBpsC1 `signature(object = "Breakpoints")`: Getter-method for the commonBpsC1 slot.

commonBpsC2<- `signature(object = "Breakpoints", value = "list")`: Setter-method for the commonBpsC2 slot.

commonBpsC2 `signature(object = "Breakpoints")`: Getter-method for the commonBpsC2 slot.

seqsC1<- `signature(object = "Breakpoints", value = "list")`: Setter-method for the seqsC1 slot.

seqsC1 `signature(object = "Breakpoints")`: Getter-method for the seqsC1 slot.

sqsC2<- `signature(object = "Breakpoints", value = "list")`: Setter-method for the seqsC2 slot.

seqsC2 `signature(object = "Breakpoints")`: Getter-method for the seqsC2 slot.

[`signature(x = "Breakpoints", i = "ANY", j = "ANY")`: Subsetting a Breakpoints object.

length `signature(x = "Breakpoints")`: Returns the number of breakpoints stored.

mergeBreakpoints `signature(breakpoints = "Breakpoints", maxDist = "missing", mergeBPs = "list")`: Merge presumably related breakpoints.

mergeBreakpoints `signature(breakpoints = "Breakpoints", maxDist = "missing", mergeBPs = "missing")`: Merge presumably related breakpoints.

mergeBreakpoints `signature(breakpoints = "Breakpoints", maxDist = "numeric", mergeBPs = "missing")`: Merge presumably related breakpoints.

names<- `signature(x = "Breakpoints", value = "ANY")`: Set the names of the breakpoints.

names `signature(x = "Breakpoints")`: Get the names of the breakpoints.

plotChimericReads `signature(brpData = "Breakpoints")`: Plot the structural variant and the chimeric reads covering its breakpoints.

summary `signature(object = "Breakpoints")`: Create a data frame summaring information about all breakpoints.

table `signature(... = "Breakpoints")`: Create a frequency table of cluster sizes.

Author(s)

Hans-Ulrich Klein, Christoph Bartenhagen

See Also

`filterChimericReads`, `detectBreakpoints`, `mergeBreakpoints`, `plotChimericReads`

MapperSet-class *Class to Contain GS Reference Mapper Output*

Description

Container to store data imported from a project of Roche's GS Reference Mapper Software. It stores all information into a Biobase ExpressionSet.

Objects from the Class

Objects can be created by calls of the form `MapperSet(filename)`. While filename is a vector containing all sample directories (i.e. directories that contain the files "mapping/454HCDiffs.txt" and "mapping/454NewblerMetrics.txt").

Slots

`assayData`**:** Object of class `AssayData`. Contains the number of reads with the respective difference and the total coverage for every variant in forward and reverse direction.

`featureData`**:** Object of class `AnnotatedDataFrame`. Contains information about the type, location and reference of each variant. If available, it shows further Ensembl variant-ids for known SNPs.

`phenoData`**:** Object of class `AnnotatedDataFrame`. By default, the phenoData contains the accession number of every sample.

`variantFilterPerc`**:** Object of class `numeric`. Contains a threshold to display only those variants, whose coverage (in percent) in forward and reverse direction in at least one sample is higher than this filter value. See `setVariantFilter` for details about setting this value.

`variantFilter`**:** Object of class `character`. Contains a vector of variant names whose coverage (in percent) in forward and reverse direction in at least one sample is higher than the filter value(s) in `variantFilterPerc`.

`dirs`**:** Object of class `character`. Based on a directory given at instantiation of the object, it contains a vector of several directories containing all relevant GS Mapper project files.

`experimentData`**:** Object of class `MIAME`. Contains details of the experiment.

`annotation`**:** Object of class `character` Label associated with the annotation package used in the experiment.

`protocolData`**:** Object of class `AnnotatedDataFrame`. Contains additional information about the samples.

`.__classVersion__`**:** Object of class `Versions`. Remembers the R and R453Toolbox version numbers used to created the MapperSet instance.

Extends

Class eSet, directly. Class VersionedBiobase, by class "eSet", distance 2. Class Versioned, by class "eSet", distance 3.

Methods

setVariantFilter(object): Sets the filter to display only those variants, whose coverage (in percent) in forward and reverse direction in at least one sample is higher than the given value.

getVariantPercentages(object) Computes the coverage for every variant over all reads (forward and/or reverse) and for each sample.

annotateVariants(object): Annotates given genomic variants. See annotateVariants for details.

htmlReport(object): Exports all (filtered) variant data into a html report. See htmlReport for details

getReadStatus(object): Reads the file "454ReadStatus.txt" in the GSM project directory which contains information about the alignment of each read (chr, pos, strand, etc.)and returns its content in a dataframe.

Author(s)

Christoph Bartenhagen

See Also

AVASet, annotateVariants, htmlReport, setVariantFilter, getVariantPercentages

Examples

```
# sum up class structure
showClass("MapperSet")

# load a GS Mapper dataset containing 3 samples and 111 variants
data(mapperSetExample)
mapperSetExample

# show contents of assay, feature and pheno data
assayData(mapperSetExample)
fData(mapperSetExample)
pData(mapperSetExample)
```

MapperSet *Creating a MapperSet*

Description

This function imports a project of Roche's GS Reference Mapper Software. It stores all information into an instance of the Biobase ExpressionSet.

Usage

```
MapperSet(dirs, samplenames)
```

Arguments

dirs	A character vector containing all sample directories (i.e. directories that contain the files "mapping/454HCDiffs.txt" (required), "mapping/454ReadStatus.txt" (optional), "mapping/454NewblerMetrics.txt"(optional)).
samplenames	A character vector containing samplenames. The order and number of samplenames must be consistent with the filenames to ensure the correctness of the MapperSet. If no samplenames are given, the filenames are used for naming.

Details

An instance of the MapperSet is derived from the Biobase eSet and thus structured into

1. assayData

variantForwCount/variantRevCount: Contain the number of reads with the respective difference in forward/reverse direction.

totalForwCount/totalRevCount: Contain the total coverage for every variant in forward/reverse direction.

2. featureData

chromosome, start/end: Give the location of each variant.

referenceBases/variantBase: Show the bases changed in each variant.

regName: The name of the region (gene) where the variant is located.

knownSNP: Lists Ensembl variant-ids for known SNPs (if any).

3. phenoData

By default, the phenoData contains the accession number of every sample.

Value

An instance of the MapperSet.

Author(s)

Christoph Bartenhagen

See Also

AVASet-class

Examples

```
# load a GS Mapper dataset containing 3 samples and 111 variants
data(mapperSetExample)
mapperSetExample
```

`alignShortReads` *Exact alignment of DNA sequences against a reference*

Description

This method aligns given sequences against a given reference genome using the `matchPDict` method. Only exact (no errors) and unique matches are returned.

Usage

```
alignShortReads(object, bsGenome, seqNames, ensemblNotation)
```

Arguments

`object`	The reads that should be aligned agiven either as a `DNAStringSet` or a `AVASet` instance. In the latter case the reference sequences are extracted and aligned.
`bsGenome`	A `bsGenome` instance providing the reference sequences.
`seqNames`	The names of the sequences in `bsGenome` that should be used. If omitted, all reference sequences are used.
`ensemblNotation`	If set to TRUE, "chr" is removed from the reference sequences' names in the returned alignment. Default value is FALSE.

Details

All reads are aligned against the reference and its reverse complement. If the reads are not in 5' to 3' orientation, they should be reversed before. Note that only exact and unique alignments are reported. Use `matchPDict` directly for more flexibility.

Value

An object of class `AlignedRead` or a `AVASet` instance.

Author(s)

Hans-Ulrich Klein

See Also

`matchPDict`, `DNAStringSet`, `AlignedRead`, `AVASet`

Examples

```
library("BSgenome.Scerevisiae.UCSC.sacCer2")
reads = DNAStringSet(c(
  "CCGTTCAAAGAGCCCTTGGCCCATAATCCACCGGTT",
  "ATCCTGCCACAGGAGTCCATGGAGGTTTCGCCA"))
alignShortReads(reads, Scerevisiae, seqNames="chrIII")
```

`annotateVariants` *Adds genomic information to variants*

Description

This method annotates given genomic variants (mutations). Annotation includes affected genes, exons and codons. Resulting amino acid changes are returned as well as dbSNP identifiers, if the mutation is already known. All information is fetched from Ensembl via biomaRt using the datasets `hsapiens_gene_ensembl` and `hsapiens_snp`.

Usage

```
annotateVariants(object, bsGenome)
```

Arguments

`object`	A data frame storing variants or an instance of AVASet/MapperSet or a data frame (see details).
`bsGenome`	An object of class `BSGenome` giving the genome to be used as reference sequence to calculate amino acid changes. This argument is only applicable when `object` is of type `MapperSet`. Default is 'BSgenome.Hsapiens.UCSC.hg19'. Note that the genome should fit to the Ensembl annotation.

Details

If a data frame is given, the following columns must be present:

start	genomic start position in the current Ensembl genome
end	genomic end position in the current Ensembl genome
chromosome	chromosome in ensembl notation (i.e. "1", "2", ..., "Y")
strand	"+" or "-" relative to the nucleotide bases given below
seqRef	reference sequence
seqMut	sequence of the observed variant
seqSur	reference sequence extended for 3 bases in both directions

The rownames of the data frame are used as mutations' names (IDs). See examples for a properly defined data drame.

Value

An object of class `AnnotatedVariants`. Affected genes, transcripts and exon as well as known SNPs are stored in a list-like structure. See the documentation of class `AnnotatedVariants-class` for details.

Author(s)

Hans-Ulrich Klein

See Also

`AnnotatedVariants-class`, `AVASet-class`, `MapperSet-class`, `htmlReport`

Examples

```
variants = data.frame(
    start=c(106157528, 106154991,106156184),
    end=c(106157528, 106154994,106156185),
    chromosome=c("4", "4", "4"),
    strand=c("+", "+", "+"),
    seqRef=c("A", "ATAG", "---"),
    seqMut=c("G", "----", "ATA"),
    seqSur=c("TACAGAA", "TTTATAGATA", "AGC---TCC"),
    stringsAsFactors=FALSE)
rownames(variants) = c("snp", "del", "ins")
## Not run: annotateVariants(variants)
```

---

assayDataAmp *Access the amplicon data of an AVASet.*

Description

Similar to `assayData` of the Biobase ExpressionSet, this function returns the assay data of the amplicon slot of an instance of the AVASet.

Usage

```
assayDataAmp(object)
```

Arguments

`object`	An `link{AVASet-class}` object.

Value

The assay data of the amplicon slot consists of a list of two data frames with the number of forward and reverse reads of all amplicons for each sample (see `AVASet-class` for details).

Author(s)

Christoph Bartenhagen

See Also

`fDataAmp`, `featureDataAmp`, `AVASet-class`

Examples

```
# load an AVA dataset containing 6 samples, 4 amplicons and 259 variants
data(avaSetExample)

# show contents of amplicon assay data
assayDataAmp(avaSetExample)
```

`avaSetExample` *Amplicon Variant Analyzer data import*

Description

This is an example of an `link{AVASet-class}` object containing the output of Roche's Amplicon Variant Analyzer Software. It consists of 6 samples, 4 amplicons and 259 variants.

Usage

```
data(avaSetExample)
```

Format

Formal class 'AVASet'

Source

'Next-generation sequencing technology reveals a characteristic pattern of molecular mutations in 72.8 leukemia by detecting frequent alterations in TET2, CBL, RAS, and RUNX1' (Kohlmann A et al., J Clin Oncol. 2010 Aug 20;28(24):3858-65. Epub 2010 Jul 19)

Examples

```
data(avaSetExample)
avaSetExample
```

`avaSetFiltered` *Amplicon Variant Analyzer data import*

Description

This is an example of an `link{AVASet-class}` object containing the output of Roche's Amplicon Variant Analyzer Software. It consists of 6 samples, 4 amplicons and 4 variants. The variants were previously filtered according to the amplicon coverage (see `setVariantFilter` for details about filtering an AVASet object).

Usage

```
data(avaSetFiltered)
```

Format

Formal class 'AVASet'

Source

'Next-generation sequencing technology reveals a characteristic pattern of molecular mutations in 72.8 leukemia by detecting frequent alterations in TET2, CBL, RAS, and RUNX1' (Kohlmann A et al., J Clin Oncol. 2010 Aug 20;28(24):3858-65. Epub 2010 Jul 19)

Examples

```
data(avaSetFiltered)
avaSetFiltered
```

---

avaSetFiltered_annot *AVASet variant annotations*

Description

These are example annotations for 4 variants of an AVASet (try `data(avaSetFiltered)` to retrieve the corresponding `link{AVASet-class}` object). The annotations include affected genes, exons and codons as well as resulting amino acid changes and dbSNP identifiers (if the mutation is already known).

Usage

```
data(avaSetFiltered_annot)
```

Format

Formal class 'AnnotatedVariants'

Source

'Next-generation sequencing technology reveals a characteristic pattern of molecular mutations in 72.8 leukemia by detecting frequent alterations in TET2, CBL, RAS, and RUNX1' (Kohlmann A et al., J Clin Oncol. 2010 Aug 20;28(24):3858-65. Epub 2010 Jul 19)

Examples

```
data(avaSetFiltered_annot)
```

---

breakpoints *Putative breakpoints of chimeric reads*

Description

This example holds two consensus (pathogenic and reciproce) breakpoints of 12 chimeric reads indicating an inversion on chromosome 16. The Breakpoints object gives access to the breakpoint locations as well as alignment information for each of the 12 reads.

Usage

```
data(breakpoints)
```

Format

Formal class 'Breakpoints'

Source

'Targeted next-generation sequencing detects point mutations, insertions, deletions, and balanced chromosomal rearrangements as well as identifies novel leukemia-specific fusion genes in a single procedure' (Leukemia, submitted)

Examples

```
data(breakpoints)
```

---

calculateTiTv *Calculate transition transversion ratio*

Description

When many point mutations are detected, the ration of transitions to transversions can be used as quality measure to assess the number of false positives.

Usage

```
  ## S4 method for signature 'AVASet'
calculateTiTv(object)
  ## S4 method for signature 'MapperSet'
calculateTiTv(object)
```

Arguments

object	An instance of AVASet or MapperSet storing the detected variants.

Details

For more information about the Ti/Tv ratio see http://www.broadinstitute.org/gsa/wiki/index.php/QC_Methods

Value

A list with two elements: A substitution matrix summarizing all observed substitutions and the transition/transversion ratio.

Author(s)

Hans-Ulrich Klein

Examples

```
data(avaSetExample)
ava = setVariantFilter(avaSetExample, c(0.03, 0.03))
calculateTiTv(ava)
```

captureArray *Custom capture array design*

Description

Design of a custom Roche NimbleGen 385k capture array. The array captures short segments corresponding to all exon regions of 92 distinct target genes (genome build hg19). In addition, contiguous genomic regions were represented for three additional genes, i.e. CBFB, MLL, and RUNX1.

Usage

```
data(captureArray)
```

Format

Formal class 'CompressedIRangesList'

Source

'Targeted next-generation sequencing detects point mutations, insertions, deletions, and balanced chromosomal rearrangements as well as identifies novel leukemia-specific fusion genes in a single procedure' (Leukemia, submitted)

Examples

```
data(captureArray)
```

convertCigar *Basic functions for CIGAR strings*

Description

These are temporary methods, that are likely to be replaced by methods from the Rsamtools package in near future.

Usage

```
extendedCIGARToList(cigars)
listToExtendedCIGAR(cigarList)
```

Arguments

cigars	A character vector with CIGAR strings.
cigarList	A list of converted CIGAR strings as produced by extendedCIGARToList

`coverageOnTarget` *Computes the coverage restricted to the target region.*

Description

This method computes the approximate coverage of each base in a given region.

Usage

```
coverageOnTarget(alnReads, targetRegion)
```

Arguments

`alnReads`	A list as returned by `scanBam` storing aligned reads.
`targetRegion`	The target region as a `RangesList`. The chromosome names must fit to the chromosome names used in the alignment information of the given reads.

Details

The detailed alignment information given by the CIGAR strings in .bam files are ignored by the function. Instead, it is assumed that the whole read alignes to the reference without indels. This is often not true for longer read (e.g. generated with Roche 454 Sequencing), but saves computation time.

Value

A list of the same length as the `alnReads` argument. Each list element is an integer vector of the same length as the target region (in bases) and stores the coverage generated by the reads from the corresponding list element of `alnReads`.

Author(s)

Hans-Ulrich Klein

See Also

`scanBam`

Examples

```
library(Rsamtools)
bamFile = system.file("extdata", "StructuralVariantDetection", "bam", "N01.bam", package=
bam = scanBam(bamFile)
region = RangesList("11"=IRanges(start=118307205, end=118395936))
cov = coverageOnTarget(bam, region)
```

demultiplexReads *Performs MID/Multiplex filtering*

Description

Roche's Genome Sequencer allows to load two or more samples on one region. To allocate sequences to samples, each sample has a unique multiplex sequence. The multiplex sequence should be the prefix of all sequences from that sample. This method demultiplexes a given set of sequences according to the given multiplex sequences (MIDs).

Usage

```
## S4 method for signature 'XStringSet,XStringSet,numeric,logical'
demultiplexReads(reads, mids, numMismatches, trim)
```

Arguments

reads	A DNAStringSet instance that contains reads starting with MIDs
mids	A DNAStringSet instance that contains the MIDs
numMismatches	The maximal number of mismatches allowed, default 2.
trim	Whether the MIDs should be cutted-out, default TRUE

Details

All given MIDs must have the same length. The algorithm computes the number of mismachtes for each MID. The read is assigned to the MID with the lowest number of mismatches. If two or more MIDs have the same number of mismachtes, or if the number of mismachtes is greater than the given argument numMismachtes, the read is not assigned to any MID. The default number of allowed mismatches is 2.

Value

demultiplexReads returns a list with one DNAStringSet instance for each MID.

Author(s)

Hans-Ulrich Klein

See Also

genomeSequencerMIDs, DNAStringSet

Examples

```
library(Biostrings)
    mids = genomeSequencerMIDs(c("MID1", "MID2", "MID3"))
    reads = DNAStringSet(c(
        paste(as.character(mids[["MID1"]]), "A", sep=""),
        paste(as.character(mids[["MID1"]]), "AA", sep=""),
        paste(as.character(mids[["MID2"]]), "AAA", sep="")))
    demultiplexReads(reads, mids)
```

`detectBreakpoints` *Clustering and consensus breakpoint detection for chimeric reads*

Description

Given a set of chimeric reads, this methods computes all putative breakpoints. First, chimeric reads are clustered such that all reads spanning the same breakpoint form a cluster. Then, a consensus breakpoint sequence and breakpoint position is computed for each cluster.

Usage

```
detectBreakpoints(chimericReads, bpDist=100, minClusterSize=4, removeSoftClips=T
```

Arguments

`chimericReads`	A list storing chimeric reads as returned by `filterChimericReads`. The list must have the format as defined by the `scanBam` method.
`bpDist`	The maximum distance in base pairs between the breakpoints of two chimeric reads at which the reads are merge to a cluster.
`minClusterSize`	Cluster whose size is below minClusterSize are be excluded from breakpoint detection.
`removeSoftClips`	If true, soft-clipped bases at the beginning or the end of a sequence are removed (see details below).
`bsGenome`	A `bsGenome` instance providing the reference sequences. If missing, the library `BSgenome.Hsapiens.UCSC.hg19` is used by default.

Details

This method is usually invoked after calling `filterChimericReads` and before calling `mergeBreakpoints`. It first forms clusters of chimeric reads (reads with exactly two local alignments) that span the same breakpoint and than computes a consensus breakpoint sequence for each cluster.

To carry out a hierarchical clustering, a measure for the distance between two chimeric reads must be defined. If reads span different chromosomes, their distance is set to infinity. The strand information of the local alignments may also indicate that two chimeric reads do not span the same breakpoint even if they span the same chromosomes. For example, the first reads has two local alignments on the positive strand whereas the second read has one local alignment on the positive strand and the other on the negative strand. In this case, the distance is set to inifinty, too. Finally, the distance measure distinguishes between the two breakpoints (sometimes called the pathogenic and the reciproce breakpoint) that originate from the same structual variant. The distance between a read from the pathogenic and a read from the reciproce breakpoint is infinity so that two different clusters will emerge. These two related breakpoints can be merge later using the `mergeBreakpoints` method. We observed that the breakpoints of these two cases often differ by a few ten or even a few hundred basepairs.

If the chromosome and strand information between two reads x and y are coherent, the Euclidian distance is used:

$$d(x,y) = (bp(x,ChrA) - bp(y,ChrA))^2 + (bp(x,ChrB) - bp(y,ChrB))^2$$

where *bp* gives the coordinates of the breakpoint for the given read and chromosome. Hierarchical clustering is applied with complete linkage and the dendrogram is cutted at a height of `bpDist` to obtain the final clusters. The `bpDist` argument does usually not influence the result, because we observed that reads spanning the same breakpoint have very little variation (only a few base pairs) in their local alignments due to sequencing errors or due to ambiguity caused by same/similar sequence of both chromosome near the breakpoint.

Although the given set of reads may belong to the same chimeric DNA, their individual breakpoints may differ in a few base pairs. Furthermore, a single read may have more than one possible breakpoint if a (small) part of the read was aligned to both parts.
The following step determines a consensus breakpoint for each cluster. It uses the supplied `bsGenome` to construct a chimeric reference sequence for all possible breakpoints over all reads within each cluster. After the reads were realigned to the chimeric reference sequences, the one that yields the highest alignment score is taken to represent best the chimeric DNA and its breakpoints.

As a preprocessing step, `detectBreakpoints` offers to remove soft clips occuring after the alignment:
Some reads may contain soft-clipped bases (e.g. linker sequences) at the beginning of the first part of the read or at the end of the second part. By default, `detectBreakpoints` removes these unaligned subsequences and adjusts the cigar string, the sequence, the sequence width (qwidth) and the local start/end coordinates.

Value

`detectBreakpoints` returns an object of class `breakpoints`, which is a list of breakpoint clusters, which gives access to all alignments and consensus breakpoints:

`seqs`	This `IRanges DataFrame` is mainly a rearranged version of the alignment input in `chimericReads`. In addition, it shows the corresponding breakpoints and local alignment coordinates.
`commonBps`	A dataframe listing the breakpoints for both parts of the chimeric reference, the associated chromosome, strand and the reference sequence itself, including positions "localStart"/"localEnd" indicating which part of the reference belongs to which breakpoint.
`commonAlign`	An object of class `PairwiseAlignedFixedSubject` of the Biostrings package that contains the alignment to the (best) consensus reference sequence.
`alignedReads`	On the basis of `commonAlign` and `commonBps`, `alignedReads` is an instance of class `AlignedRead` containing all aligned reads including their associated chromosomes, strands, and positions. Since the reference is a chimeric sequence each read has two chromosome and two strand entries.

Author(s)

Hans-Ulrich Klein, Christoph Bartenhagen

See Also

`filterChimericReads` `mergeBreakpoints` `plotChimericReads`

Examples

```
# Construct a small example with three chimeric reads
# (=6 local alignments) in bam format as given by
# aligners such as BWA-SW.
# The first two reads originate from the same case but
# from different strands. The third read originate from
# the reciprocal breakpoint.
library("BSgenome.Scerevisiae.UCSC.sacCer2")
bamReads = list()
bamReads[[1]] = list(
    qname=c("seq1", "seq1", "seq2", "seq2", "seq3", "seq3"),
    flag = as.integer(c(0, 0, 16, 16, 0, 0)),
    rname = factor(c("II", "III", "III", "II", "III", "II")),
    strand = factor(c("+", "+", "-", "-", "+", "+")),
    pos = as.integer(c(99951, 200000, 200000, 99951, 199950, 100001)),
    qwidth = as.integer(c(100, 100, 100, 100, 100, 100)),
    cigar = c("50M50S","50S50M","50S50M","50M50S","50M50S", "50S50M"),
    seq = DNAStringSet(c(
        paste(substr(Scerevisiae$chrII, start=99951, stop=100000),
              substr(Scerevisiae$chrIII, start=200000, stop=200049),
              sep=""),
        paste(substr(Scerevisiae$chrII, start=99951, stop=100000),
              substr(Scerevisiae$chrIII, start=200000, stop=200049),
              sep=""),
        paste(substr(Scerevisiae$chrIII, start=200000, stop=200049),
              substr(Scerevisiae$chrII, start=99951, stop=100000),
              sep=""),
        paste(substr(Scerevisiae$chrIII, start=200000, stop=200049),
              substr(Scerevisiae$chrII, start=99951, stop=100000),
              sep=""),
        paste(substr(Scerevisiae$chrIII, start=199950, stop=199999),
              substr(Scerevisiae$chrII, start=100001, stop=100050),
              sep=""),
        paste(substr(Scerevisiae$chrIII, start=199950, stop=199999),
              substr(Scerevisiae$chrII, start=100001, stop=100050),
              sep="")))
)

bps = detectBreakpoints(bamReads, minClusterSize=1, bsGenome=Scerevisiae)
summary(bps)
table(bps)

mergeBreakpoints(bps)
```

`fDataAmp` *Access the amplicon data of an AVASet.*

Description

Similar to `fData` of the Biobase ExpressionSet, this function returns the feature data of the amplicon slot of an instance of the AVASet.

Usage

```
fDataAmp(object)
```

Arguments

object	An link{AVASet-class} object.

Value

The feature data of the amplicon slot contains the names, primers, start/end positions and reference sequences of all amplicons (seeAVASet-class for details). It returns a data frame.

Author(s)

Christoph Bartenhagen

See Also

featureDataAmp, assayDataAmp,AVASet-class

Examples

```
# load an AVA dataset containing 6 samples, 4 amplicons and 259 variants
data(avaSetExample)
avaSetExample

# show contents amplicon feature data
fDataAmp(avaSetExample)
```

---

featureDataAmp *Access the amplicon data of an AVASet*

Description

Similar to featureData of the Biobase ExpressionSet, this function returns the feature data and feature meta of the amplicon slot of an instance of the AVASet.

Usage

```
featureDataAmp(object)
```

Arguments

object	An link{AVASet-class} object.

Value

The feature data of the amplicon slot contains the names, primers, start/end positions and reference sequences of all amplicons (see AVASet-class for details). The returned object is of class AnnotatedDataFrame.

Author(s)

Christoph Bartenhagen

See Also

fDataAmp, assayDataAmp, AVASet-class,

Examples

```
# load an AVA dataset containing 6 samples, 4 amplicons and 259 variants
data(avaSetExample)
avaSetExample

# show contents amplicon feature data
featureDataAmp(avaSetExample)
```

---

filterChimericReads

Extract chimeric reads and apply filtering steps to remove artificial

---

Description

Chimeric reads may be caused by sequencing a chromosomal aberration or by technical issues during sample preparation. This method implements several filter steps to remove false chimeric reads.

Usage

```
  ## S4 method for signature 'list,RangesList,DNAString,numeric,numeric'
filterChimericReads(alnReads, targetRegion, linkerSeq, minDist, dupReadDist)
```

Arguments

alnReads	A list storing the aligned reads as produced by the function scanBam.
targetRegion	A object of class rangesList containing the target region of e.g. a used capture array. The parameter may be omitted in case of a non targeted sequencing approach.
linkerSeq	A linker sequence that was used during sample preparation. It may be omitted.
minDist	The minimum distance between two local alignments (see details), default 1000
dupReadDist	The maximum distance between the 5 prime start position of two duplicated reads (see details), default 1.

Details

The following filter steps are performed:

1. All chimeric reads with exactly two local alignments are extracted. Reads with more than two local alignments are discarded.

2. If the targetRegion argument is given, chimeric reads must have one local alignment at least overlapping the the target region. If both local alignments are outside the target region, the read is discarded.

3. If the linkerSeq argument is given, all chimeric reads that have the linker sequence between their local alignments are removed. When searching the linker sequence, 4 mismatches or indels are

allowed and the linker sequence must not start or end within the first or last ten bases of the read. The function searches for the linkerSeq and for it's reverse complement.

4. Two local alignment of a read must have minDist reads between the alignments (if both alignment are on the same chromosome). Otherwise, the read seems to span a deletion and not a chromosomal aberration and is discarded.

5. Duplicated reads are removed. Two reads are duplicated, if the lie on the same strand and have the same 5 prime start position. Due to sequencing and alignment errors, the start position may vary for a maximum of dupReadDist bases. In case of duplicated reads, only the longest read is kept.

Reads passing all filtering steps are returned in the `list` structure as given by the `alnReads` argument (as derived from the `scanBam` method). A data frame with information about the number of reads that passed each filter is added to the list.

Value

A `list` containing only filtered chimeric reads. The `list` has the same structure like the given argument `alnReads`. Additionally, one element "log" with logging information of each filtering step is added.

Author(s)

Hans-Ulrich Klein

See Also

`detectBreakpoints`, `mergeBreakpoints`, `Breakpoints-class`, `scanBam`, `sequenceCaptureLinke`

Examples

```
library(Rsamtools)
bamFile = system.file("extdata", "StructuralVariantDetection", "bam", "N01.bam", package=
bam = scanBam(bamFile)
data(captureArray)
linker = sequenceCaptureLinkers("gSel3")[[1]]
filterReads = filterChimericReads(bam, targetRegion=captureArray, linkerSeq=linker)
```

`genomeSequencerMIDs`

Retrieve GS multiplex sequences

Description

This method returns the standard multiplex sequences used by the Genome Sequence MID library kits.

Usage

```
## S4 method for signature 'missing'
genomeSequencerMIDs()
       ## S4 method for signature 'character'
genomeSequencerMIDs(mid)
```

Arguments

mid	Character vector with multiplex sequences' IDs (MIDs)

Details

If the argument mid is omitted, all 14 available multiplex sequences are returned.

Value

genomeSequencerMIDs returns a DNAStringSet with the requested multiplex sequences.

Author(s)

Hans-Ulrich Klein

See Also

demultiplexReads

Examples

```
genomeSequencerMIDs()
genomeSequencerMIDs(c("MID1", "MID3"))
```

---

getAlignedReads *Import reads from an Amplicon Variant Analyzer project*

Description

For a given AVASet, this function imports all aligned reads belonging to all (or some selected) amplicons of all samples.

Usage

```
getAlignedReads(object, amplicons, dir)
```

Arguments

object	An instance of the link{AVASet-class}.
amplicons	An (optional) character vector of amplicon names as mentioned in the amplicon feature data (see fDataAmp).
dir	Usually, the method tries to retrieve the path to the AVA project from the given link{AVASet} object. However, if it fails to find the directory, dir can be used to set the root directory of the AVA project.

Details

This function reports all reads for all samples together. If you want to get the reads for some samples individually, try subsetting your AVASet as in the examples below.

Value

One DNAStringSet that contains all aligned reads for all samples (eventually restricted to some given amplicons).

Author(s)

Christoph Bartenhagen

See Also

AVASet-class, fDataAmp

Examples

```
# load an AVA dataset containing 6 samples, 4 amplicons and 259 variants
data(avaSetExample)

# import all reads for amplicon "TET2_E11.04" of the first sample
avaProjectDir = system.file("extdata", "AVASet", package = "R453Plus1Toolbox")
alnReads = getAlignedReads(avaSetExample[, 1], dir=avaProjectDir, amplicons="TET2_E11.04"
show(alnReads)
```

---

getAminoAbbr *Get amino acid abbreviations*

Description

This function returns a table with amino acid names as first column and the according abbreviations as row names.

Usage

```
getAminoAbbr()
```

---

getVariantPercentages *Variant coverage*

Description

This function computes the coverage for each variant (in forward and/or reverse direction) for all samples. The coverage is defined as the percentual amount of reads that cover a variant.

Usage

```
getVariantPercentages(object, direction="both")
```

Arguments

object	An instance of class AVASet-class or MapperSet-class.
direction	A character indicating the direction ("forward", "reverse" or "both").

Details

If the direction was set to "both", the percentages are computed over the sum of both directions. Otherwise it is computed only over the occurences in one direction (forward or reverse). The occurences can be accesses via assayData.

Value

getVariantPercentages returns a data frame with all percentages/frequencies for all samples.

Author(s)

Christoph Bartenhagen

See Also

setVariantFilter.

Examples

```
# load a (filtered) AVA dataset containing 6 samples, 4 amplicons and 4 variants
data(avaSetFiltered)
avaSetFiltered

# both directions
getVariantPercentages(avaSetFiltered, direction="both")
# this is equivalent to
(assayData(avaSetFiltered)[[1]] + assayData(avaSetFiltered)[[3]]) / (assayData(avaSetFilt

# forward direction only
getVariantPercentages(avaSetFiltered, direction="forward")
# this is equivalent to
assayData(avaSetFiltered)[[1]] / assayData(avaSetFiltered)[[2]]

# reverse direction only
getVariantPercentages(avaSetFiltered, direction="reverse")
# this is equivalent to
assayData(avaSetFiltered)[[3]] / assayData(avaSetFiltered)[[4]]
```

htmlReport *HTML-Report Builder for the AVASet and MapperSet*

Description

This function creates a HTML variant and quality report for a given AVASet or MapperSet instance.

Usage

```
htmlReport(object, annot, blocks=c(), transcripts=c(), sampleCols, minMut=3, dir
```

Arguments

object	An AVASet-class or MapperSet-class instance.
annot	An instance of class AnnotatedVariants you get by calling annotateVariants. If no such argument is supplied the data will be read from the Ensembl database automatically for all variants.
blocks	Character vector of block names for each variant. The variants will then be structured into several blocks. If no such list is supplied, the report consists of just one (big) table for all variants.
transcripts	Character vector containing Ensembl transcript-IDs that order the according entries on the transcript pages. Transcripts given in this argument will appear on top of the transcript page.
sampleCols	Character vector of column names of the sample data (phenoData) of the AVASet/MapperSet object to filter the sample output on the transcript pages. All columns will be listed if no such argument is given.
minMut	If the value of minMut is greater than zero, the report lists only variants, whose coverage for at least one sample is higher than minMut (percentage between 0 and 100).
dir	Character with the desired output directory. By defaultm the directory "HTML-Report" will be created in the current directory.
title	Heading for the first page with the variant information.

Details

The report is structured into two (MapperSet) or three (AVASet) parts containing variant and quality information:

- The main page sums up given variant information like the name, type, reference gene, position (see fData, annotateVariants).
  Using the argument blocks, the main page can be individually structured by assigning a block name to each variant.
  The main page can be further structured by samples. For a given AVASet object, every sample links to another short quality report showing only the amplicon coverage for this sample.
- Every variant on the main page links to a page with further details about the affected genes and transcripts (e.g. Ensembl gene-IDs, transcript-IDs, codon sequences, changes of amino acids (if coding)).
- Only in case of AVASet object: A quality report shows the coverage of every amplicon in forward and/or reverse direction. Further plots display the coverage by MID and PTP (if this information is given in the pheno data of the object).

Author(s)

Christoph Bartenhagen, Hans-Ulrich Klein, Christian Ruckert

See Also

annotateVariants.

Examples

```
# note: all examples save the report to the directory "htmlReportExample" in your current

# load a filtered AVA dataset containing 6 samples, 4 amplicons and 4 variants
# and its variant annotations
data("avaSetFiltered")
data("avaSetFiltered_annot")

# create a full report showing all (unfiltered) information
htmlReport(avaSetFiltered, avaSetFiltered_annot, dir="htmlReportExample", title="htmlRepo
# create a report that emphasizes on samples with variants covered by at least 50% of the
htmlReport(avaSetFiltered, avaSetFiltered_annot, dir="htmlReportExample", title="htmlRepo

# create a report that is structured by the reference genes
library("ShortRead")
refs = sapply(fData(avaSetFiltered)$referenceSeq, function(x)
    subset(pData(alignData(referenceSequences(avaSetFiltered))), pData(alignData(referenc
htmlReport(avaSetFiltered, avaSetFiltered_annot, dir="htmlReportExample", title="htmlRepo

# create a report whose sample information only lists the sample ids
pData(avaSetFiltered)
sampleCols = "SampleID"
htmlReport(avaSetFiltered, avaSetFiltered_annot, dir="htmlReportExample", title="htmlRepo
```

---

mapperSetExample *GS Reference Mapper data import*

Description

This is an example of an link{MapperSet-class} object containing the output of Roche's GS Reference Mapper Software. It consists of 3 samples and 111 variants.

Usage

```
data(mapperSetExample)
```

Format

Formal class 'MapperSet'

Source

'Targeted next-generation sequencing detects point mutations, insertions, deletions, and balanced chromosomal rearrangements as well as identifies novel leukemia-specific fusion genes in a single procedure' (Leukemia, submitted)

Examples

```
data(mapperSetExample)
mapperSetExample
```

`mergeBreakpoints` *Identify and merge related breakpoints caused by the same variant.*

Description

Structural variation like transversions or inversion cause two breakpoints. In the context of fusion genes, these are called the pathogenic breakpoint and the reciproce breakpoint. The method `detectBreakpoints` processes each breakpoint individually and does explicitly not put reads from the pathogenic and reciproce breakpoint into the same cluster. Hence, it is usually sensible to call this methods afterward to search for related pairs of breakpoints to gain more confidence about the existence of a structural variation.

Usage

```
mergeBreakpoints(breakpoints, maxDist, mergeBPs)
```

Arguments

`breakpoints`	An object of class `Breakpoints` storing the breakpoints that will (potentially) be merged.
`maxDist`	The maximal distance in basepairs at which two breakpoints will be merged. Default value is 1000.
`mergeBPs`	An optional list of vectors of length two giving the breakpoints that should be merged. If this argument is given, the method will not search for related breakpoints.

Details

If the `maxDist` argument is given, the method compares each pair of breakpoints and checks, whether the two breakpoints may belong to the same structural variation. In addition to the spanned chromosomes, the orientation and the strand information of the reads spanning the breakpoints are also compared for this purpose. If chrosmosome, orientation and strand information of two breakpoints go well together, they will be merged, if the absolute distance of the breakpoints on chromosome A plus the absolute distance on chromosome B is smaller or equal to `maxDist`. If one breakpoint has more than one potential mate breakpoint for merging, it will be merged with the first candidate and a warning message is printed. The default value of `maxDist` is 1000.\

If the `mergeBPs` argument is given, the method will not search for related breakpoints but simply merge the given breakpoints. `mergeBPs` must be a `list` with vectors of length two that either contain the names of the indices of the breakpoints that should be merged.\

The arguments `maxDist` and `mergeBPs` cannot be given together. The given `Breakpoints` object must not contain breakpoints that have been merged before.

Value

An object of class `Breakpoints` storing merged and unmerged breakpoints.

Author(s)

Hans-Ulrich Klein

See Also

detectBreakpoints, Breakpoints-class, plotChimericReads

Examples

```
# Load bam file and filter chimeric reads
library(Rsamtools)
bamFile = system.file("extdata", "StructuralVariantDetection", "bam", "N01.bam", package=
bam = scanBam(bamFile)
data(captureArray)
linker = sequenceCaptureLinkers("gSel3")[[1]]
filterReads = filterChimericReads(bam, targetRegion=captureArray, linkerSeq=linker)

# detect breakpoints of size >= 3
breakpoints = detectBreakpoints(filterReads, minClusterSize=3)
table(breakpoints)
summary(breakpoints)

# merge breakpoints
breakpoints = mergeBreakpoints(breakpoints)
summary(breakpoints)
```

---

plotAmpliconCoverage

Creates a plot visualizing the number of reads per amplicon

Description

A function for visualizing the number of reads per amplicon or per MID / pico titer plate.

Usage

```
## S4 method for signature 'AVASet,character,logical'
plotAmpliconCoverage(avaSet, type="amplicon", bothDirections=TRUE, cex.names=0.8
```

Arguments

avaSet	An instance of AVASet.
type	A character vector specifying the type of plot.
bothDirections	A logical value determining whether the plot sums forward and reverse reads or shows them separately.
cex.names	Font size of the amplicon name labels.
cex.axis	Font size of axes' labels.
las	Orientation of amplicon name labels.
col	Colors used in the plot.
...	Arguments to be passed to methods, such as graphical parameters (see 'par').

Details

If the argumnet type is "amplicon", the number of reads for each amplicon are visualized. In case of a AVASet with one sample, a barplot with one bar for each amplicon is created. In case of more than one sample, a boxplot with one box for each amplicon is plotted. If type is "mid", a boxplot with one box for each MID is created. If type is "ptp", a boxplot with one box for each pico titer plate is created.

Author(s)

Hans-Ulrich Klein

See Also

AVASet

Examples

```
## Not run: data(avaSetExample)
plotAmpliconCoverage(avaSetExample)
plotAmpliconCoverage(avaSetExample[,1])
## End(Not run)
```

---

plotChimericReads *Plots chimeric reads*

Description

This function plots a given set of aligned chimeric reads along a reference sequence. It plots the breakpoints of translations or inversions and marks deletions, insertions and mismatches. Optionally, it displays all base pairs in a given region around the breakpoint.

Usage

```
plotChimericReads(brpData, geneSymbols=FALSE, plotMut=TRUE, plotBasePairs=FALSE,
    col=c("red", "green", "black", "orange"))
```

Arguments

brpData	A Breakpoints object containing the consensus breakpoint of all reads and the consensus reference sequence as returned by the methods detectBreakpoints and mergeBreakpoints. Since only one plot is made, the function will only work for objects of class Breakpoints having length one.
geneSymbols	Boolean value whether to automatically load and plot the gene symbols from the Ensembl database. Additionally, geneSymbols can be a vector of two strings for an own annotation.
plotMut	Boolean value whether to mark deletions, insertions and mismatches.
plotBasePairs	Optionally, plotChimericReads displays all base pairs in a given region around the breakpoint (see maxBasePairs).
maxBasePairs	The maximum number of base pairs to be plotted. Only used in conjunction with plotBasePairs=TRUE.

legend	A logical value (TRUE/FALSE) whether to plot a legend that explains the colouration of the insertions, deletions, mismatches and breakpoints.
title	A title for the plot.
col	A vector of four colours to draw insertions, deletions, mismatches and breakpoints. In this order, the default colours are "red", "green", "black" and "orange" (use `colours()` to see a list of possible values).

Details

This method is intended to be run after the pipeline for structural variant detection. Therefore, see the methods `filterChimericReads`, `detectBreakpoints` and `mergeBreakpoints` to correctly preprocess your alignment before running `plotChimericReads`.

Note

It is recommended to first create and resize the output device (e.g. the plotting window or a pdf file) before plotting. For example, on Unix systems you may try `X11(width=w, height=h)` or `pdf(file="plotChimericReads.pdf", width=w, height=h)` for some window width w (e.g. w=12) and window height h (e.g. h=6).

Author(s)

Christoph Bartenhagen

See Also

`Breakpoints-class`, `detectBreakpoints`, `mergeBreakpoints`

Examples

```
# load breakpoint data containing twelve chimeric reads describing an inversion in chromo
data("breakpoints")
breakpoints

# standard plot
# (only arrangement of reads plotted; breakpoints in orange, deletions
# in red, insertions in green and mismatches in black by default)
plotChimericReads(breakpoints)

# plot base pairs in the breakpoint region (+/- 32bp)
## Not run: plotChimericReads(breakpoints, plotBasePairs=TRUE, maxBasePairs=32)

# use custom colours and display a legend:
# deletions="brown", insertions="blue", mismatches="yellow", breakpoints="gray"
plotChimericReads(breakpoints, col=c("brown", "blue", "yellow", "gray"), legend=TRUE)
```

`plotVariants` *Plots variant positions*

Description

This function illustrates the positions and types of all variants within a given transcript.

Usage

```
plotVariants(varData, transcript, legend=TRUE, regions=c(), regLabel="(region la
```

Arguments

`varData`	An instance of class `annotatedVariants` you get by calling `annotateVariants`. If no such argument is supplied the data will be read from the Ensembl database automatically for all variants.
`transcript`	This argument can either be a string containing an Ensembl transcript-id or a data frame containing the associated Ensembl transcript information. In the latter case, the data frame requires the columns "ensembl_transcript_id", "rank", "cds_start", "cds_end" and "cds_length". If only the Ensembl transcript-id is passed, the function will return an appropriate data frame.
`legend`	A logical value (TRUE/FALSE) whether to plot a legend that explains the type of mutation.
`regions`	A vector with transcript positions (start and end for every region) to be highlighted in the plot. It must have the form `regions=c(start1, end1, start2, end2, start3, end3 ...)`.
`regLabel`	If `legend=TRUE` and `regions` were specified, it is recommended to give a short label that explains the highlighted region(s).
`col`	A vector specifying four colours for missense point mutations, silent point mutations, nonsense point mutations and deletions (in this order). By default, these mutations are drawn as grey, white, black arrows and grey diamonds respectively.
`title`	A title for the plot.

Details

The plot shows only coding regions and distinguishes between missense, silent and nonsense point mutations and deletions.

Value

If an Ensembl transcript-id is passed as `transcript` argument, the function will return a data frame containing the Ensembl transcript information required for plotting ("ensembl_transcript_id", "rank", "cds_start", "cds_end" and "cds_length"). This data frame may prove useful for future plots with the same transcript.
If such a data frame is passed directly, the function returns nothing.

Author(s)

Christoph Bartenhagen

See Also

annotateVariants.

Examples

```
# one missense, one nonsense point mutation and one deletion
variants = data.frame(
    row.names=c("missense", "deletion", "nonsense"),
    start=c(106157528, 106157635, 106193892),
    end=c(106157528, 106157635, 106193892),
    chromosome=c("4", "4", "4"),
    strand=c("+", "+", "+"),
    seqRef=c("A", "G", "C"),
    seqMut=c("G", "-", "T"),
    seqSur=c("TACAGAA", "TAAGCAG", "CGGCGAA"),
    stringsAsFactors=FALSE)

# annotate variants with affected genes, exons and codons (may take a minute to finish)
## Not run: varAnnot = annotateVariants(variants)

# plot variants vor transcript "ENST00000380013" and mark region within [700, 1000]
## Not run: plotVariants(varAnnot, transcript="ENST00000380013", legend=TRUE,regions=c(70
```

---

plotVariationFrequency

Create an AVA style variation frequency plot

Description

This method creates a plot similar to the variation frequency plot in Roche's GS Amplicon Variant Analyzer. The plot shows the reference sequence along the x-axis and indicates variants as bars at the appropriate positions. The height of the bars corresponds to the percentage of reads carrying the variant. A second y-axis indicates the absolute number of reads covering the variant.

Usage

```
plotVariationFrequency(object, plotRange, ...)
```

Arguments

object	A character pointing to an Amplicon Variant Analyser Global Alignmnet export file.
plotRange	A two dimensional numeric vector giving the start and end base of the reference sequence that should be plotted.
...	Arguments passed to other plotting methods. Especially, argument col: Optional character vector of length 7 specifying the bars' colors indicating different base substitutions or deletions. See details. And argument sequenceCex: Optional numeric value specifying the size of the reference sequence's bases.

Details

The text file used as imput must have the format generated by the AVA export function. Such a file can be generated using the export button in the Global Alignment view of the AVA software. The `col` argument specifies the colours used for different bases and deletions. The following listing gives the meaning of the i-th position of the col vector (default values in braces):

1. A (green)
2. C (blue)
3. G (black)
4. T (red)
5. N (purple)
6. deletion (gray)

Author(s)

Hans-Ulrich Klein

Examples

```
## Not run:
file = system.file("extdata", "AVAVarFreqExport", "AVAVarFreqExport.xls", package="R453Pl
plotVariationFrequency(file, plotRange=c(50, 150))
## End(Not run)
```

`readsOnTarget` *Check for each read whether it aligns within the given region.*

Description

This methods checks (approximately) whether the given reads align within a given region.

Usage

```
readsOnTarget(alnReads, targetRegion)
```

Arguments

`alnReads`	A list as returned by `scanBam` storing aligned reads.
`targetRegion`	The target region as a `RangesList`. The chromosome names must fit to the chromosome names used in the alignment information of the given reads.

Details

The detailed alignment information given by the CIGAR strings in .bam files are ignored by the function. Instead, it is assumed that the whole read alignes to the reference without indels. This is often not true for longer read (e.g. generated with Roche 454 Sequencing), but saves computation time. Hence, this method is useful to approximate the number of reads that align in the target region of a targeted sequencing experiment.

Value

A list with one logical vector for each list entry in alnReads. The logical vector indicates for each read whether it overlaps with at least one base from any target region or not.

Author(s)

Hans-Ulrich Klein

See Also

scanBam

Examples

```
library(Rsamtools)
bamFile = system.file("extdata", "StructuralVariantDetection", "bam", "N01.bam", package=
bam = scanBam(bamFile)
region = RangesList("11"=IRanges(start=118307205, end=118395936))
targetReads = readsOnTarget(bam, region)
sum(targetReads[[1]])
```

---

referenceSequences *Access the reference sequences of an AVASet*

Description

This function give access to a slot of an instance of the AVASet storing information about all reference sequences of the amplicons.

Usage

```
referenceSequences(object)
```

Arguments

object	An link{AVASet-class} object.

Value

The data is stored in an object of class AlignedRead and thus gives information about all reference sequences and their position on a chromosome (if alignShortReads has been called before).

Author(s)

Christoph Bartenhagen

See Also

alignShortReads

Examples

```
# load an AVA dataset containing 6 samples, 4 amplicons and 259 variants
data(avaSetExample)

referenceSequences(avaSetExample)
```

`removeLinker` *Remove linker sequences located at the start of short reads*

Description

If linkers are attached during sample preparation, it may be useful to remove the linkers' sequences after sequencing. This method finds and removes linker sequences that are located at the start of the given reads.

Usage

```
## S4 method for signature 'XStringSet,DNAString,logical,numeric,numeric'
removeLinker(reads, linker, removeReadsWithoutLinker, minOverlap, penalty)
```

Arguments

`reads`	A `DNAStringSet` instance that contains reads possibly having linkers at their start site
`linker`	A `DNAString` instance with the linker's sequence
`removeReadsWithoutLinker`	Whether reads without linkers should be removed. Default is FALSE
`minOverlap`	The minimal score that must be achived when aligning the linker. Default is length(linker)/2
`penalty`	The penalty for substitutions or indels. Default is -2

Details

The best alignment of the linker within the start (length of linker + 5) of each given sequence is computed. The followong scoring schema is used: Each matching bases scores +1. Each substitution or indel scores the given penalty argument (default: penalty=-2). There are no penalties for gaps and the end of the linker (overlap). An alignment is considered as match, if the scores is larger of equal to minOverlap (default: minOverlap=round(length(linker)/2)). In cases of a successful match, the subsequence from position 1 until the end of the linker's alignment is removed.

Value

`removeLinker` returns a `DNAStringSet` with trimmed reads.

Author(s)

Hans-Ulrich Klein

See Also

sequenceCaptureLinkers, DNAStringSet, pairwiseAlignment

Examples

```
linker = sequenceCaptureLinkers()[[1]]
reads = DNAStringSet(c(
    "CTCGAGAATTCTGGATCCTCAAA",
         "GAATTCTGGATCCTCAAA",
    "CTCGAGAAAAAAAAATCCTCAAA"))
removeLinker(reads, linker)
```

---

sequenceCaptureLinkers

Retrieve NimbleGen's sequence capture linkers

Description

This method returns the NimbleGen's linker sequences used with their sequence capture arrays. See pp.29-30 in the NimbleGen Arrays User's Guide.

Usage

```
## S4 method for signature 'character'
sequenceCaptureLinkers(name)
       ## S4 method for signature 'missing'
sequenceCaptureLinkers()
```

Arguments

name	Character vector with linker sequences' names

Details

If the argument name is omitted, both linker sequences are returned.

Value

sequenceCaptureLinkers returns a DNAStringSet with the requested linker sequences.

Author(s)

Hans-Ulrich Klein

See Also

removeLinker

Examples

```
sequenceCaptureLinkers()
```

setVariantFilter *Filters output of variant information*

Description

This functions sets the filter to display only those variants, whose amplicon coverage (in percent) in forward and reverse direction in at least one sample is higher than a given value. The coverage is defined as the percentual amount of reads that cover a variant.

Usage

```
setVariantFilter(object, filter=0)
```

Arguments

object	An instance of an link{AVASet-class} or MapperSet-class.
filter	A filter value between 0 and 1. If two values are given in a vector, the variants are filterd according to the forward (first value) and reverse direction (second value) separately. In this case, a variant has to meet both requirements.

Details

Setting the filter affects the assayData and the featureData of the variant slot. See also getVariantPercentages for further details.

Value

setVariantFilter returns the given link{AVASet-class}/link{MapperSet-class} instance with an updated filter value.

Author(s)

Christoph Bartenhagen

See Also

link{AVASet-class}, link{MapperSet-class}, getVariantPercentages.

Examples

```
# load an AVA dataset containing 6 samples, 4 amplicons and 259 variants
data(avaSetExample)
avaSetExample

# use only those variants that are covered by at least 10% of all reads in one sample in
avaSetExample = setVariantFilter(avaSetExample, filter=0.1)
avaSetExample

# use only those variants that are covered by at least 0.1% of all reads in one sample in
# and by at least 0% in reverse direction (259 -> 6 variants)
avaSetExample = setVariantFilter(avaSetExample, filter=c(0.1, 0))
avaSetExample
```

```
# reset filter values to zero
avaSetExample = setVariantFilter(avaSetExample, filter=0)
# or simply
avaSetExample = setVariantFilter(avaSetExample)
```

`transcriptdf` *Ensembl transcript information for variant plotting*

Description

This example contains Ensembl transcript information (here for transcript 'ENST00000380013') as created, used and returned by the function `link{plotVariants}`. It mainly holds locations of coding areas within the transcript that will then be highlighted in the variant plot.

Usage

```
data(transcriptdf)
```

Format

A data frame with 9 observations on the following 5 variables.

`ensembl_transcript_id` Ensembl id for the transcript

`rank` exon rank

`cds_start` start of a codon

`cds_end` end of a codon

`cds_length` length of a codon

Examples

```
data(transcriptdf)
```

Index