

# Package ‘customProDB’

November 14, 2024

**Type** Package

**Title** Generate customized protein database from NGS data, with a focus on RNA-Seq data, for proteomics search

**Version** 1.46.0

**Date** 2024-04-03

**Author** Xiaojing Wang

**Maintainer** Xiaojing Wang <xwang.research@gmail.com>  
Bo Wen <wenbostar@gmail.com>

**Description** Database search is the most widely used approach for peptide and protein identification in mass spectrometry-based proteomics studies. Our previous study showed that sample-specific protein databases derived from RNA-Seq data can better approximate the real protein pools in the samples and thus improve protein identification. More importantly, single nucleotide variations, short insertion and deletions and novel junctions identified from RNA-Seq data make protein database more complete and sample-specific. Here, we report an R package customProDB that enables the easy generation of customized databases from RNA-Seq data for proteomics search. This work bridges genomics and proteomics studies and facilitates cross-omics data integration.

**License** Artistic-2.0

**Depends** R (>= 3.0.1), IRanges, AnnotationDbi, biomaRt(>= 2.17.1)

**Imports** S4Vectors (>= 0.9.25), DBI, GenomeInfoDb, GenomicRanges, Rsamtools (>= 1.10.2), GenomicAlignments, Biostrings (>= 2.26.3), GenomicFeatures (>= 1.32.0), stringr, RCurl, plyr, VariantAnnotation (>= 1.13.44), rtracklayer, RSQLite, txdbmaker, AhoCorasickTrie, methods

**Suggests** RMariaDB, BSgenome.Hsapiens.UCSC.hg19

**LazyLoad** yes

**biocViews** ImmunoOncology, Sequencing, MassSpectrometry, Proteomics, SNP, RNASeq, Software, Transcription, AlternativeSplicing, FunctionalGenomics

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

**git\_branch** RELEASE\_3\_20

**git\_last\_commit** 585e069

**git\_last\_commit\_date** 2024-10-29

**Repository** Bioconductor 3.20

**Date/Publication** 2024-11-14

## Contents

aaVariation . . . . .	2
Bed2Range . . . . .	3
calculateRPKM . . . . .	4
easyRun . . . . .	5
easyRun_mul . . . . .	6
InputVcf . . . . .	7
JunctionType . . . . .	8
Multiple_VCF . . . . .	9
Outputaberrant . . . . .	10
OutputNovelJun . . . . .	11
Outputproseq . . . . .	12
OutputsharedPro . . . . .	13
OutputVarprocodingseq . . . . .	14
OutputVarproseq . . . . .	15
OutputVarproseq_single . . . . .	16
Positionincoding . . . . .	17
PrepareAnnotationEnsembl . . . . .	18
PrepareAnnotationRefseq . . . . .	20
SharedJunc . . . . .	21
Varlocation . . . . .	22
<b>Index</b>	<b>23</b>

---

aaVariation	<i>get the functional consequence of SNVs located in coding region</i>
-------------	--

---

## Description

Variations can be divided into SNVs and INDELS. By taking the output of positionincoding() as input, aaVariation() function predicts the consequences of SNVs in the harbored transcript, such as synonymous or non-synonymous.

## Usage

```
aaVariation(position_tab, coding, ...)
```

## Arguments

position_tab	a data frame from Positionincoding()
coding	a data frame cotaining coding sequence for each protein.
...	Additional arguments

## Details

this function predicts the consequence for SNVs. for INDELS, use Outputaberrant().

**Value**

a data frame containing consequence for each variations.

**Author(s)**

Xiaojing Wang

**Examples**

```
vcffile <- system.file("extdata/vcfs", "test1.vcf", package="customProDB")
vcf <- InputVcf(vcffile)
table(values(vcf[[1]])[['INDEL']])

index <- which(values(vcf[[1]])[['INDEL']]==FALSE)
SNVvcf <- vcf[[1]][index]
load(system.file("extdata/refseq", "exon_anno.RData", package="customProDB"))
load(system.file("extdata/refseq", "dbsnpinCoding.RData", package="customProDB"))
load(system.file("extdata/refseq", "procodingseq.RData", package="customProDB"))
postable_snv <- PositioninCoding(SNVvcf,exon,dbsnpinCoding)
txlist <- unique(postable_snv[, 'txid'])
codingseq <- procodingseq[procodingseq[, 'tx_id'] %in% txlist,]
mtab <- aaVariation (postable_snv,codingseq)
mtab[1:3,]
```

---

Bed2Range

*Generate a GRanges objects from BED file.*

---

**Description**

Read BED file into a GRanges object. This function requires complete BED file. Go to <https://genome.ucsc.edu/FAQ/FAQ> for more information about BED format.

**Usage**

```
Bed2Range(bedfile, skip = 1, covfilter = 5, ...)
```

**Arguments**

bedfile	a character contains the path and name of a BED file.
skip	the number of lines of the BED file to skip before beginning to read data, default 1.
covfilter	the number of minimum coverage for the candidate junction, default 5.
...	additional arguments

**Details**

Read BED file contain junctions into a GRanges object.

**Value**

a GRanges object containing all candidate junctions from the BED file.

**Author(s)**

Xiaojing Wang

**Examples**

```
bedfile <- system.file("extdata/beds", "junctions1.bed", package="customProDB")
jun <- Bed2Range(bedfile, skip=1,covfilter=5)
length(jun)
```

---

 calculateRPKM

---

*Calculate RPKM for each transcripts based on exon read counts.*


---

**Description**

Normalized expression level based on exon read counts. The default output is a vector containing RPKMs for each transcript. vector name is the transcript name. calculate the RPKMs by chromosome. If `proteinCodingonly=TRUE`, vector name will be set to protein name, and only output RPKMs for the protein coding transcripts.

**Usage**

```
calculateRPKM(bamFile, exon, proteinCodingonly = TRUE,
  ids = NULL, ...)
```

**Arguments**

<code>bamFile</code>	a the input BAM file name.
<code>exon</code>	a dataframe of exon annotations.
<code>proteinCodingonly</code>	if TRUE only output RPKMs for protein coding transcripts, the name of output vector will be protein id. if FALSE, output the RPKM for all transcripts.
<code>ids</code>	a dataframe containing gene/transcript/protein id mapping information.
<code>...</code>	additional arguments

**Details**

calculate RPKM from a BAM file based on exon read counts

**Value**

RPKM value for all transcripts or protein coding transcripts.

**Author(s)**

Xiaojing Wang

**Examples**

```
##test1.bam file is part of the whole bam file.
load(system.file("extdata/refseq", "exon_anno.RData", package="customProDB"))
bamFile <- system.file("extdata/bams", "test1_sort.bam", package="customProDB")
load(system.file("extdata/refseq", "ids.RData", package="customProDB"))
RPKM <- calculateRPKM(bamFile,exon,proteinCodingonly=TRUE,ids)
```

---

easyRun	<i>An integrated function to generate customized protein database for a single sample</i>
---------	---

---

### Description

Generate a customized protein database for a single sample.

### Usage

```
easyRun(bamFile, RPKM = NULL, vcfFile, annotation_path,
        outfile_path, outfile_name, rpkm_cutoff = 1,
        INDEL = FALSE, lablersid = FALSE, COSMIC = FALSE,
        nov_junction = FALSE, bedFile = NULL, genome = NULL,
        ...)
```

### Arguments

bamFile	Input BAM file name
RPKM	Alternative to bamFile,default NULL, a vector containing expression level for proteins. (e.g. FPKMs from cufflinks)
vcfFile	Input VCF file name.
outfile_path	Folder path for the output FASTA files.
outfile_name	Output FASTA file name.
annotation_path	The path of saved annotation.
rpkm_cutoff	The cutoff of RPKM value. see 'cutoff' in function Outputproseq for more detail.
INDEL	If the vcfFile contains the short insertion/deletion. Default is FALSE.
lablersid	If includes the dbSNP rsid in the header of each sequence, default is FALSE. Users should provide dbSNP information when running function Positionincoding() if put TRUE here.
COSMIC	If output the cosmic ids in the variation table.Default is FALSE. If choose TRUE, there must have cosmic.RData in the annotation folder.
nov_junction	If output the peptides that cover novel junction into the database. if TRUE, there should be splicemax.RData in the annotation folder.
bedFile	The path of bed file which contains the splice junctions identified in RNA-Seq.
genome	A BSgenome object(e.g. Hsapiens). Default is NULL.
...	Additional arguments

### Details

The function gives a more convenient way for proteomics researchers to generate customized database for a single sample.

### Value

A table file contains detailed variation information and several FASTA files.

**Author(s)**

Xiaojing Wang

**Examples**

```

bamFile <- system.file("extdata/bams", "test1_sort.bam",
                      package="customProDB")
vcffile <- system.file("extdata/vcfs", "test1.vcf", package="customProDB")
annotation_path <- system.file("extdata/refseq", package="customProDB")
outfile_path <- tempdir()
outfile_name <- 'test'

easyRun(bamFile, RPKM=NULL, vcffile, annotation_path, outfile_path,
        outfile_name, rpkm_cutoff=1, INDEL=TRUE, lablersid=TRUE,
        COSMIC=TRUE, nov_junction=FALSE)

```

---

easyRun_mul	<i>An integrated function to generate consensus protein database from multiple samples</i>
-------------	--

---

**Description**

Generate consensus protein database for multiple samples in a single function.

**Usage**

```

easyRun_mul(bamFile_path, RPKM_mtx = NULL, vcffile_path,
            annotation_path, rpkm_cutoff, share_num = 2,
            var_shar_num = 2, outfile_path, outfile_name,
            INDEL = FALSE, lablersid = FALSE, COSMIC = FALSE,
            nov_junction = FALSE, bedFile_path = NULL,
            genome = NULL, junc_shar_num = 2, ...)

```

**Arguments**

bamFile_path	The path of BAM files
RPKM_mtx	Alternative to bamFile_path, default NULL, a matrix containing expression level for proteins in each sample. (e.g. FPKMs from cufflinks)
vcffile_path	The path of VCF files
annotation_path	The path of already saved annotation, which will be used in the function
rpkm_cutoff	Cutoffs of RPKM values. see 'cutoff' in function OutputsharedPro for more information
share_num	The minimum share sample numbers for proteins which pass the cutoff.
var_shar_num	Minimum sample number of recurrent variations.
outfile_path	The path of output FASTA file
outfile_name	The name prefix of output FASTA file
INDEL	If the vcffile contains the short insertion/deletion. Default is FALSE.

lablrsid	If includes the dbSNP rsid in the header of each sequence, default is FALSE. Users should provide dbSNP information when running function Positionincoding() if put TRUE here.
COSMIC	If output the cosmic ids in the variation table. Default is FALSE. If choose TRUE, there must have cosmic.RData in the annotation folder.
nov_junction	If output the peptides that cover novel junction into the database. if TRUE, there should be splicemax.RData in the annotation folder.
bedFile_path	The path of BED files which contains the splice junctions identified in RNA-Seq.
genome	A BSgenome object(e.g. Hsapiens). Default is NULL. Required if nov_junction==TRUE.
junc_shar_num	Minimum sample number of recurrent splicing junctions.
...	Additional arguments

### Details

The function give a more convenient way for proteomics researchers to generate customized database of multiple samples.

### Value

A table file contains detailed variation information and several FASTA files.

### Author(s)

Xiaojing Wang

### Examples

```
bampath <- system.file("extdata/bams", package="customProDB")
vcfFile_path <- system.file("extdata/vcfs", package="customProDB")
annotation_path <- system.file("extdata/refseq", package="customProDB")
outfile_path <- tempdir()
outfile_name <- 'mult'

easyRun_mul(bampath, RPKM_mtx=NULL, vcfFile_path, annotation_path, rpkm_cutoff=1,
            share_num=2, var_shar_num=2, outfile_path, outfile_name, INDEL=TRUE,
            lablrsid=TRUE, COSMIC=TRUE, nov_junction=FALSE)
```

---

InputVcf

*Generate a list of GRanges objects from a VCF file.*

---

### Description

The InputVcf() function generates a list of GRanges object from a single VCF file.

### Usage

```
InputVcf(vcfFile, ...)
```

**Arguments**

vcffile            a character contains the path and name of a VCF file  
 ...                additional arguments

**Details**

Read all fields in a VCF file into GRanges object.

**Value**

a list of GRanges object containing a representation of data from the VCF file

**Author(s)**

Xiaojing Wang

**Examples**

```
## multiple samples in one VCF file

vcffile <- system.file("extdata", "test_mul.vcf", package="customProDB")
vcfs <- InputVcf(vcffile)
length(vcfs)

## single sample

vcffile <- system.file("extdata/vcfs", "test1.vcf", package="customProDB")
vcf <- InputVcf(vcffile)
length(vcf)
```

---

JunctionType            *Annotates the junctions in a bed file.*

---

**Description**

For identified splice junctions from RNA-Seq, this function finds the junction types for each entry according to the given annotation. Six types of junctions are classified. find more details in the tutorial.

**Usage**

```
JunctionType(jun, splicemax, txdb, ids, ...)
```

**Arguments**

jun                a GRange object for junctions, the output of function Bed2Range.  
 splicemax        a known exon splice matrix from the annotation.  
 txdb             a TxDb object.  
 ids              a dataframe containing gene/transcript/protein id mapping information.  
 ...              additional arguments



**Details**

Go to <https://genome.ucsc.edu/FAQ/FAQformat.html#format1> for more information about BED format.

**Value**

a data frame of type and source for each junction.

**Author(s)**

Xiaojing Wang

**Examples**

```
bedfile <- system.file("extdata/beds", "junctions1.bed", package="customProDB")
jun <- Bed2Range(bedfile, skip=1, covfilter=5)
load(system.file("extdata/refseq", "splicemax.RData", package="customProDB"))
load(system.file("extdata/refseq", "ids.RData", package="customProDB"))
txdb <- loadDb(system.file("extdata/refseq", "txdb.sqlite",
                          package="customProDB"))
junction_type <- JunctionType(jun, splicemax, txdb, ids)
table(junction_type[, 'jun_type'])
```

---

Multiple\_VCF

*Generate shared variation dataset from multiple VCF files*

---

**Description**

Load multiple vcf files and output a GRange object with SNVs present in multiple samples.

**Usage**

```
Multiple_VCF(vcfs, share_num, ...)
```

**Arguments**

vcfs	a list of GRanges object which input from multiple VCF files using function InputVcf.
share_num	Two options, percentage format or sample number.
...	additional arguments

**Details**

This function allows to limit SNVs that are present in at least m out of n VCF files.

**Value**

a GRange object that contains the shared variations

**Author(s)**

Xiaojing Wang

**Examples**

```
path <- system.file("extdata/vcfs", package="customProDB")
vcfFiles<- paste(path, '/', list.files(path, pattern="*vcf$"), sep='')
vcfs <- lapply(vcfFiles, function(x) InputVcf(x))
shared <- Multiple_VCF(vcfs, share_num=2)
```

---

Outputaberrant                      *generate FASTA file containing short INDEL*

---

**Description**

Short insertion/deletion may lead to aberrant proteins in cells. We provide a function to generate FASTA file containing this kind of proteins.

**Usage**

```
Outputaberrant(positiontab, outfile, coding, proteinseq,
               ids, RPKM = NULL, ...)
```

**Arguments**

positiontab	a data frame which is the output of function Positionincoding() for INDELS.
outfile	output file name
coding	a data frame cotaining coding sequence for each protein.
proteinseq	a data frame cotaining amino acid sequence for each protein.
ids	a dataframe containing gene/transcript/protein id mapping information.
RPKM	if includes the RPKM value in the header of each sequence, default is NULL.
...	Additional arguments.

**Details**

the function applys the INDEL into the coding sequence, then translates them into protein sequence, terminated by stop codon. Remove the sequences the same as normal ones or as part of normal ones.

**Value**

FASTA file containing aberrant proteins.

**Author(s)**

Xiaojing Wang

**Examples**

```
vcffile <- system.file("extdata/vcfs", "test1.vcf", package="customProDB")
vcf <- InputVcf(vcffile)
table(values(vcf[[1]])[['INDEL']])
index <- which(values(vcf[[1]])[['INDEL']] == TRUE)
indelvcf <- vcf[[1]][index]

load(system.file("extdata/refseq", "exon_anno.RData", package="customProDB"))
load(system.file("extdata/refseq", "dbsnpinCoding.RData",
  package="customProDB"))
load(system.file("extdata/refseq", "procodingseq.RData",
  package="customProDB"))
load(system.file("extdata/refseq", "proseq.RData", package="customProDB"))
load(system.file("extdata/refseq", "ids.RData", package="customProDB"))
postable_indel <- PositioninCoding(indelvcf, exon)
txlist_indel <- unique(postable_indel[, 'txid'])
codingseq_indel <- procodingseq[procodingseq[, 'tx_id'] %in% txlist_indel, ]
outfile <- paste(tempdir(), '/test_indel.fasta', sep='')
Outputaberrant(postable_indel, coding=codingseq_indel,
  proteinseq=proteinseq, outfile=outfile, ids=ids)
```

---

OutputNovelJun

*generate peptide FASTA file that contains novel junctions.*


---

**Description**

Three-frame translation of novel junctions. And remove those could be found in normal protein sequences. This function requires a genome built by BSgenome package.

**Usage**

```
OutputNovelJun(junction_type, genome, outfile,
  proteinseq, ...)
```

**Arguments**

junction_type	a data frame which is the output of function JunctionType()
genome	a BSgenome object. (e.g. Hsapiens)
outfile	output file name
proteinseq	a data frame containing amino acid sequence for each protein.
...	Additional arguments.

**Value**

FASTA file that contains novel junction peptides.

**Author(s)**

Xiaojing Wang

**Examples**

```

bedfile <- system.file("extdata/beds", "junctions1.bed", package="customProDB")
jun <- Bed2Range(bedfile,skip=1,covfilter=5)
load(system.file("extdata/refseq", "splicemax.RData", package="customProDB"))
load(system.file("extdata/refseq", "ids.RData", package="customProDB"))
txdb <- loadDb(system.file("extdata/refseq", "txdb.sqlite",
                          package="customProDB"))
junction_type <- JunctionType(jun, splicemax, txdb, ids)
table(junction_type[, 'jun_type'])
chrom <- paste('chr',c(1:22,'X','Y','M'),sep='')
junction_type <- subset(junction_type, seqnames %in% chrom)
outf_junc <- paste(tempdir(), '/test_junc.fasta', sep='')
load(system.file("extdata/refseq", "proseq.RData", package="customProDB"))
library('BSgenome.Hsapiens.UCSC.hg19')
OutputNovelJun <- OutputNovelJun(junction_type, Hsapiens, outf_junc,
                                proteinseq)

```

---

Outputproseq	<i>output FASTA format file contains proteins that have expression level above the cutoff</i>
--------------	---

---

**Description**

Get the FASTA file of proteins that pass RPKM cutoff. the FASTA ID line contains protein ID, gene ID, HGNC symbol and description

**Usage**

```
Outputproseq(rpkm, cutoff = "30%", proteinseq, outfile,
            ids, ...)
```

**Arguments**

rpkm	a numeric vector containing RPKM for each protein
cutoff	cutoff of RPKM value. Two options are available, percentage format or RPKM. By default we use "30 proteins according to their RPKMs.
proteinseq	a dataframe containing protein ids and protein sequences.
outfile	output file name.
ids	a dataframe containing gene/transcript/protein id mapping information.
...	additional arguments

**Details**

by taking the RPKM value as input, the function outputs sequences of the proteins that pass the cutoff.

**Value**

FASTA file contains proteins with RPKM above the cutoff.

**Author(s)**

Xiaojing Wang

**Examples**

```
load(system.file("extdata/refseq", "exon_anno.RData", package="customProDB"))
load(system.file("extdata/refseq", "proseq.RData", package="customProDB"))
bamFile <- system.file("extdata/bams", "test1_sort.bam",
  package="customProDB")
load(system.file("extdata/refseq", "ids.RData", package="customProDB"))
RPKM <- calculateRPKM(bamFile, exon, proteincodingonly=TRUE, ids)
outf1 <- paste(tempdir(), '/test_rpkm.fasta', sep='')
Outputproseq(RPKM, 1, proteinseq, outf1, ids)
```

---

OutputsharedPro	<i>Output the sequences of proteins with high expressions in multiple samples.</i>
-----------------	--

---

**Description**

Output a FASTA file containing shared proteins with expression above cutoff in multiple samples

**Usage**

```
OutputsharedPro(RPKMs, cutoff = "30%",
  share_sample = "50%", proteinseq, outfile, ids, ...)
```

**Arguments**

RPKM	RPKM matrix; row name (protein name) is required.
cutoff	a percentage format cutoff (e.g. '30' a vector with each element as a vlaue cutoff referring to one sample
share_sample	the minimum share sample numbers for proteins which pass the cutoff.
proteinseq	a dataframe containing protein ids and protein sequences
outfile	output file name
ids	a dataframe containing gene/transcript/protein id mapping information.
...	additional arguments

**Details**

this function takes RPKM matrix as input, users can set two paramteters,cutoff and shared, to generated a consensus expressed database

**Value**

a FASTA file containing proteins with RPKM above the cutoff in at least certain number of samples

**Author(s)**

Xiaojing Wang

**Examples**

```

path <- system.file("extdata/bams", package="customProDB")
load(system.file("extdata/refseq", "exon_anno.RData", package="customProDB"))
load(system.file("extdata/refseq", "proseq.RData", package="customProDB"))
load(system.file("extdata/refseq", "ids.RData", package="customProDB"))
bamFile<- paste(path, '/', list.files(path, pattern="*bam$"), sep='')
rpkms <- sapply(bamFile,function(x)
  calculateRPKM(x, exon, proteincodingonly=TRUE, ids))
outfile <- paste(tempdir(), '/test_rpkm_share.fasta', sep='')
OutputsharedPro(rpkms, cutoff=1, share_sample=2, proteinq,
  outfile, ids)

```

---

OutputVarprocodingseq *Output the variant(SNVs) protein coding sequences*

---

**Description**

Output 'snvprocoding'

**Usage**

```
OutputVarprocodingseq(vartable, procodingseq, ids, lablersid = FALSE, ...)
```

**Arguments**

vartable	A data frame which is the output of aaVariation().
procodingseq	A dataframe containing protein ids and coding sequence for the protein.
ids	A dataframe containing gene/transcript/protein id mapping information.
lablersid	If includes the dbSNP rsid in the header of each sequence, default is FALSE. Must provide dbSNP information in function Positionincoding() if put TRUE here.
...	Additional arguments

**Details**

This function uses the output of aaVariation() as input, introduces the nonsynonymous variation into the protein database.

**Value**

a data frame containing protein coding sequence proteins with single nucleotide variation.

**Author(s)**

Xiaojing Wang

**Examples**

```
vcffile <- system.file("extdata/vcfs", "test1.vcf", package="customProDB")
vcf <- InputVcf(vcffile)
table(values(vcf[[1]])[['INDEL']])
index <- which(values(vcf[[1]])[['INDEL']] == FALSE)
SNVvcf <- vcf[[1]][index]
load(system.file("extdata/refseq", "exon_anno.RData",
package="customProDB"))
load(system.file("extdata/refseq", "dbsnpinCoding.RData",
package="customProDB"))
load(system.file("extdata/refseq", "procodingseq.RData",
package="customProDB"))
load(system.file("extdata/refseq", "ids.RData", package="customProDB"))
load(system.file("extdata/refseq", "proseq.RData", package="customProDB"))
postable_snv <- PositioninCoding(SNVvcf, exon, dbsnpinCoding)
txlist <- unique(postable_snv[, 'txid'])
codingseq <- procodingseq[procodingseq[, 'tx_id'] %in% txlist, ]
mtab <- aaVariation (postable_snv, codingseq)
OutputVarprocodingseq(mtab, codingseq, ids, lablersid=TRUE)
```

---

OutputVarproseq

*Output the variant(SNVs) protein sequences into FASTA format*


---

**Description**

Output the non-synonymous SNVs into FASTA file.

**Usage**

```
OutputVarproseq(vartable, proteinq, outfile, ids, lablersid = FALSE,
RPKM = NULL, ...)
```

**Arguments**

vartable	A data frame which is the output of aaVariation().
proteinq	A dataframe containing protein ids and the protein sequence.
outfile	Output file name.
ids	A dataframe containing gene/transcript/protein id mapping information.
lablersid	If includes the dbSNP rsid in the header of each sequence, default is FALSE. Must provide dbSNP information in function PositioninCoding() if put TRUE here.
RPKM	If includes the RPKM value in the header of each sequence, default is NULL.
...	Additional arguments

**Details**

This function uses the output of aaVariation() as input, introduces the nonsynonymous variation into the protein database.

**Value**

a FASTA file and a data frame containing proteins with single nucleotide variation.

**Author(s)**

Xiaojing Wang

**Examples**

```
vcffile <- system.file("extdata/vcfs", "test1.vcf", package="customProDB")
vcf <- InputVcf(vcffile)
table(values(vcf[[1]])[['INDEL']])
index <- which(values(vcf[[1]])[['INDEL']] == FALSE)
SNVvcf <- vcf[[1]][index]
load(system.file("extdata/refseq", "exon_anno.RData",
package="customProDB"))
load(system.file("extdata/refseq", "dbsnpinCoding.RData",
package="customProDB"))
load(system.file("extdata/refseq", "procodingseq.RData",
package="customProDB"))
load(system.file("extdata/refseq", "ids.RData", package="customProDB"))
load(system.file("extdata/refseq", "proseq.RData", package="customProDB"))
postable_snv <- PositioninCoding(SNVvcf, exon, dbsnpinCoding)
txlist <- unique(postable_snv[, 'txid'])
codingseq <- procodingseq[procodingseq[, 'tx_id'] %in% txlist, ]
mtab <- aaVariation (postable_snv, codingseq)
outfile <- paste(tempdir(), '/test_snv.fasta', sep='')
snvproseq <- OutputVarproseq(mtab, proteinseq, outfile, ids, lablersid=TRUE, RPKM=NULL)
```

---

OutputVarproseq\_single

*Output the variant(SNVs) protein sequences into FASTA format*

---

**Description**

Output the non-synonymous SNVs into FASTA file, one SNV per sequence.

**Usage**

```
OutputVarproseq_single(vartable, proteinseq, outfile,
ids, lablersid = FALSE, RPKM = NULL, ...)
```

**Arguments**

vartable	A data frame which is the output of aaVariation().
proteinseq	A dataframe containing protein ids and the protein sequence.
outfile	Output file name.
ids	A dataframe containing gene/transcript/protein id mapping information.
lablersid	If includes the dbSNP rsid in the header of each sequence, default is FALSE. Must provide dbSNP information in function PositioninCoding() if put TRUE here.



RPKM                    If includes the RPKM value in the header of each sequence. default is NULL.  
 ...                    Additional arguments

### Details

This function uses the output of aaVariation() as input, introduces the nonsynonymous variation into the protein database. If a protein have more than one SNVs, introduce one SNV each time, end up with equal number of sequences.

### Value

FASTA file containing proteins with single nucleotide variation.

### Author(s)

Xiaojing Wang

### Examples

```
vcffile <- system.file("extdata/vcfs", "test1.vcf", package="customProDB")
vcf <- InputVcf(vcffile)
table(values(vcf[[1]])[['INDEL']])
index <- which(values(vcf[[1]])[['INDEL']] == FALSE)
SNVvcf <- vcf[[1]][index]
load(system.file("extdata/refseq", "exon_anno.RData",
package="customProDB"))
load(system.file("extdata/refseq", "dbsnpinCoding.RData",
package="customProDB"))
load(system.file("extdata/refseq", "procodingseq.RData",
package="customProDB"))
load(system.file("extdata/refseq", "ids.RData", package="customProDB"))
load(system.file("extdata/refseq", "proseq.RData", package="customProDB"))
postable_snv <- Positionincoding(SNVvcf, exon, dbsnpinCoding)
txlist <- unique(postable_snv[, 'txid'])
codingseq <- procodingseq[procodingseq[, 'tx_id'] %in% txlist, ]
mtab <- aaVariation (postable_snv, codingseq)
outfile <- paste(tempdir(), '/test_snv_single.fasta', sep='')
OutputVarproseq_single(mtab, proteineseq, outfile, ids, lablersid=TRUE)
```

---

Positionincoding                    *Find the position in coding sequence for each variation.*

---

### Description

For those variations labeled with "Coding", positionincoding() function computes the position of variation in the coding sequence of each transcript.

### Usage

```
Positionincoding(Vars, exon, dbsnp = NULL, COSMIC = NULL,
  ...)
```

**Arguments**

Vars	a GRanges object of variations
exon	a dataframe of exon annotations for protein coding transcripts.
dbsnp	provide a GRanges object of known dbsnp information to include dbsnp evidence into the output table, default is NULL.
COSMIC	provide a GRanges object of known COSMIC information to include COSMIC evidence into the output table, default is NULL.
...	additional arguments

**Details**

this function prepares input data frame for aaVariation().

**Value**

a data frame containing the position in coding sequence for each variation

**Author(s)**

Xiaojing Wang

**Examples**

```
vcffile <- system.file("extdata/vcfs", "test1.vcf", package="customProDB")
vcf <- InputVcf(vcffile)
table(values(vcf[[1]])[['INDEL']])
index <- which(values(vcf[[1]])[['INDEL']] == TRUE)
indelvcf <- vcf[[1]][index]

index <- which(values(vcf[[1]])[['INDEL']] == FALSE)
SNVvcf <- vcf[[1]][index]
load(system.file("extdata/refseq", "exon_anno.RData",
  package="customProDB"))
load(system.file("extdata/refseq", "dbsnpinCoding.RData",
  package="customProDB"))
load(system.file("extdata/refseq", "procodingseq.RData",
  package="customProDB"))
load(system.file("extdata/refseq", "cosmic.RData",
  package="customProDB"))
postable_snv <- PositioninCoding(SNVvcf, exon, dbsnpinCoding, COSMIC=cosmic)
```

---

PrepareAnnotationEnsembl

*prepare annotation from ENSEMBL*

---

**Description**

prepare the annotation from ENSEMBL through biomaRt.

## Usage

```
PrepareAnnotationEnsembl(mart, annotation_path, splice_matrix = FALSE,  
  dbsnp = NULL, transcript_ids = NULL, COSMIC = FALSE, ...)
```

## Arguments

<code>mart</code>	which version of ENSEMBL dataset to use. see useMart from package biomaRt for more detail.
<code>annotation_path</code>	specify a folder to store all the annotations
<code>splice_matrix</code>	whether generate a known exon splice matrix from the annotation. this is not necessary if you don't want to analyse junction results, default is FALSE.
<code>dbsnp</code>	specify a snp dataset you want to use for the SNP annotation, default is NULL.
<code>transcript_ids</code>	optionally, only retrieve transcript annotation data for the specified set of transcript ids
<code>COSMIC</code>	whether to download COSMIC data, default is FALSE.
<code>...</code>	additional arguments

## Details

this function automatically prepares all annotation information needed in the following analysis.

## Value

several .RData file containing annotations needed for following analysis.

## Author(s)

Xiaojing Wang

## Examples

```
ensembl <- useEnsembl(biomart = 'genes',  
  dataset = 'hsapiens_gene_ensembl',  
  version = 111)  
  
annotation_path <- tempdir()  
transcript_ids <- c("ENST00000234420", "ENST00000269305", "ENST00000445888",  
  "ENST00000257430", "ENST00000508376", "ENST00000288602",  
  "ENST00000269571", "ENST00000256078", "ENST00000384871")  
  
PrepareAnnotationEnsembl(mart=ensembl, annotation_path=annotation_path,  
  splice_matrix=FALSE, dbsnp=NULL, transcript_ids=transcript_ids,  
  COSMIC=FALSE)
```

---

```
PrepareAnnotationRefseq
```

*prepare annotation for Refseq*

---

## Description

prepare the annotation for Refseq through UCSC table browser.

## Usage

```
PrepareAnnotationRefseq(genome = "hg19", CDSfasta, pepfasta, annotation_path,
  dbsnp = NULL, transcript_ids = NULL, splice_matrix = FALSE,
  ClinVar = FALSE, ...)
```

## Arguments

genome	specify the UCSC DB identifier (e.g. "hg19")
CDSfasta	path to the fasta file of coding sequence.
pepfasta	path to the fasta file of protein sequence, check 'introduction' for more detail.
annotation_path	specify a folder to store all the annotations.
dbsnp	specify a snp dataset to be used for the SNP annotation, default is NULL. (e.g. "snp148")
transcript_ids	optionally, only retrieve transcript annotation data for the specified set of transcript ids. Default is NULL.
splice_matrix	whether generate a known exon splice matrix from the annotation. this is not necessary if you don't want to analyse junction results, default is FALSE.
ClinVar	whether to download ClinVar data, default is FALSE.
...	additional arguments

## Value

several .RData file containing annotations needed for further analysis.

## Author(s)

Xiaojing Wang

## Examples

```
## Not run:

transcript_ids <- c("NM_001126112", "NM_033360", "NR_073499", "NM_004448",
  "NM_000179", "NR_029605", "NM_004333", "NM_001127511")
pepfasta <- system.file("extdata", "refseq_pro_seq.fasta",
  package="customProDB")
CDSfasta <- system.file("extdata", "refseq_coding_seq.fasta",
  package="customProDB")
annotation_path <- tempdir()
PrepareAnnotationRefseq(genome='hg38', CDSfasta, pepfasta, annotation_path,
```

```
dbsnp=NULL, transcript_ids=transcript_ids,  
splice_matrix=FALSE, ClinVar=FALSE)
```

```
## End(Not run)
```

---

SharedJunc

*Generate shared junctions dataset from multiple BED files*

---

## Description

Load multiple BED files and output a GRange object with junctions present in multiple samples.

## Usage

```
SharedJunc(juns, share_num = 2, ...)
```

## Arguments

juns	a list of GRanges object which input from multiple VCF files using function InputVcf.
share_num	Junctions must occurs in this number of samples to be consider. Two options, percentage format or sample number.
...	additional arguments

## Details

This function allows to limit junctions that are present in at least m out of n BED files.

## Value

a GRange object that contains the shared junctions

## Author(s)

Xiaojing Wang

## Examples

```
path <- system.file("extdata/beds", package="customProDB")  
bedFiles<- paste(path, '/', list.files(path, pattern="*bed$"), sep='')  
juncs <- lapply(bedFiles, function(x) Bed2Range(x, skip=1, covfilter=5))  
shared <- SharedJunc(juncs, share_num=2)  
shared
```

---

Varlocation	<i>Annotates the variations with genomic location.</i>
-------------	--

---

### Description

For a given GRange object of variations, the Varlocation() function finds the genomic locations for each entry according to the given annotation. Seven labels are used to describe the location (intergenic, intro\_nonProcoding, exon\_nonProcoding, intron, 5utr, 3utr and coding). details of the definition can be found in the tutorial.

### Usage

```
Varlocation(Vars, txdb, ids, ...)
```

### Arguments

Vars	a GRange object of variations
txdb	a TxDb object.
ids	a dataframe containing gene/transcript/protein id mapping information
...	additional arguments

### Details

see 'introduction' for more details

### Value

a data frame of locations for each variation

### Author(s)

Xiaoqing Wang

### Examples

```
## Not run:
vcffile <- system.file("extdata/vcfs", "test1.vcf", package="customProDB")
vcf <- InputVcf(vcffile)

table(values(vcf[[1]])[['INDEL']])
index <- which(values(vcf[[1]])[['INDEL']] == TRUE)
indelvcf <- vcf[[1]][index]

index <- which(values(vcf[[1]])[['INDEL']] == FALSE)
SNVvcf <- vcf[[1]][index]

txdb <- loadDb(system.file("extdata/refseq", "txdb.sqlite", package="customProDB"))
load(system.file("extdata/refseq", "ids.RData", package="customProDB"))
SNVloc <- Varlocation(SNVvcf, txdb, ids)
indelloc <- Varlocation(indelvcf, txdb, ids)
table(SNVloc[, 'location'])

## End(Not run)
```

# Index

[aaVariation](#), [2](#)

[Bed2Range](#), [3](#)

[calculateRPKM](#), [4](#)

[easyRun](#), [5](#)

[easyRun\\_mul](#), [6](#)

[InputVcf](#), [7](#)

[JunctionType](#), [8](#)

[Multiple\\_VCF](#), [9](#)

[Outputaberrant](#), [10](#)

[OutputNovelJun](#), [11](#)

[Outputproseq](#), [12](#)

[OutputsharedPro](#), [13](#)

[OutputVarprocodingseq](#), [14](#)

[OutputVarproseq](#), [15](#)

[OutputVarproseq\\_single](#), [16](#)

[Positionincoding](#), [17](#)

[PrepareAnnotationEnsembl](#), [18](#)

[PrepareAnnotationRefseq](#), [20](#)

[SharedJunc](#), [21](#)

[Varlocation](#), [22](#)