

# Package ‘ORFik’

January 16, 2021

**Type** Package

**Title** Open Reading Frames in Genomics

**Version** 1.10.2

**Encoding** UTF-8

**Description**

R package for analysis of transcript and translation features through manipulation of sequence data and NGS data like Ribo-Seq, RNA-Seq, TCP-

Seq and CAGE. It is generalized in the sense that any transcript region can be analysed, as the name hints to it was made with investigation of ribosomal patterns over Open Reading Frames (ORFs) as it's primary use case.

ORFik is extremely fast through use of C++, data.table and GenomicRanges.

Package allows to reassign starts of the transcripts with the use of CAGE-Seq data, automatic shifting of RiboSeq reads, finding of Open Reading Frames for whole genomes and much more.

**biocViews** ImmunoOncology, Software, Sequencing, RiboSeq, RNASeq, FunctionalGenomics, Coverage, Alignment, DataImport

**License** MIT + file LICENSE

**LazyData** TRUE

**BugReports** <https://github.com/Roleren/ORFik/issues>

**URL** <https://github.com/Roleren/ORFik>

**Depends** R (>= 3.6.0), IRanges (>= 2.17.1), GenomicRanges (>= 1.35.1), GenomicAlignments (>= 1.19.0)

**Imports** S4Vectors (>= 0.21.3), GenomeInfoDb (>= 1.15.5), GenomicFeatures (>= 1.31.10), AnnotationDbi (>= 1.45.0), rtracklayer (>= 1.43.0), Rcpp (>= 1.0.0), data.table (>= 1.11.8), Biostrings (>= 2.51.1), biomartr, Rsamtools (>= 1.35.0), BiocGenerics (>= 0.29.1), BiocParallel (>= 1.19.0), SummarizedExperiment (>= 1.14.0), DESeq2 (>= 1.24.0), ggplot2 (>= 2.2.1), gridExtra (>= 2.3), cowplot (>= 1.0.0), GGally (>= 1.4.0), methods (>= 3.6.0), R.utils, utils, stats, fst (>= 0.9.2), xml2 (>= 1.2.0), tools

**RoxygenNote** 7.1.1

**Suggests** testthat, rmarkdown, knitr, BiocStyle, BSgenome, BSgenome.Hsapiens.UCSC.hg19

**LinkingTo** Rcpp

**VignetteBuilder** knitr

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

**git\_branch** RELEASE\_3\_12

**git\_last\_commit** 564b6b5

**git\_last\_commit\_date** 2021-01-15

**Date/Publication** 2021-01-15

**Author** Haakon Tjeldnes [aut, cre, dtc],  
Kornel Labun [aut, cph],  
Katarzyna Chyzynska [ctb, dtc],  
Yamila Torres Cleuren [ctb, ths],  
Evind Valen [ths, fnd]

**Maintainer** Haakon Tjeldnes <hauken\_heyken@hotmail.com>

## R topics documented:

ORFik-package . . . . .	7
addCdsOnLeaderEnds . . . . .	8
addNewTSSOnLeaders . . . . .	9
allFeaturesHelper . . . . .	9
artificial.orfs . . . . .	10
assignAnnotations . . . . .	11
assignFirstExonsStartSite . . . . .	12
assignLastExonsStopSite . . . . .	13
assignTSSByCage . . . . .	13
asTX . . . . .	15
bamVarName . . . . .	16
bamVarNamePicker . . . . .	17
bedToGR . . . . .	18
cellLineNames . . . . .	18
changePointAnalysis . . . . .	19
checkRFP . . . . .	20
checkRNA . . . . .	20
codonSumsPerGroup . . . . .	21
collapse.by.scores . . . . .	21
collapse.fastq . . . . .	22
collapseDuplicatedReads . . . . .	23
collapseDuplicatedReads,GAlignmentPairs-method . . . . .	23
collapseDuplicatedReads,GAlignments-method . . . . .	24
collapseDuplicatedReads,GRanges-method . . . . .	25
combn.pairs . . . . .	26
computeFeatures . . . . .	26
computeFeaturesCage . . . . .	28
conditionNames . . . . .	30
convertLibs . . . . .	30
convertToOneBasedRanges . . . . .	32
correlation.plots . . . . .	33
countOverlapsW . . . . .	34
countTable . . . . .	35
countTable_regions . . . . .	36

coverageByTranscriptW . . . . .	37
coverageGroupings . . . . .	38
coverageHeatMap . . . . .	38
coveragePerTiling . . . . .	40
coverageScorings . . . . .	42
create.experiment . . . . .	43
defineIsoform . . . . .	45
defineTrailer . . . . .	46
detectRibosomeShifts . . . . .	47
disengagementScore . . . . .	49
distToCds . . . . .	50
distToTSS . . . . .	51
download.SRA . . . . .	52
download.SRA.metadata . . . . .	53
downstreamFromPerGroup . . . . .	54
downstreamN . . . . .	55
downstreamOfPerGroup . . . . .	55
DTEG.analysis . . . . .	56
DTEG.plot . . . . .	57
entropy . . . . .	58
experiment-class . . . . .	59
experiment.colors . . . . .	61
export.bed12 . . . . .	62
export.bedo . . . . .	63
export.bedoc . . . . .	63
export.ofst . . . . .	64
export.ofst,GAlignmentPairs-method . . . . .	65
export.ofst,GAlignments-method . . . . .	66
export.ofst,GRanges-method . . . . .	67
export.wiggle . . . . .	68
extendLeaders . . . . .	69
extendsTSSexons . . . . .	70
extendTrailers . . . . .	70
filepath . . . . .	71
filterCage . . . . .	72
filterExtremePeakGenes . . . . .	73
filterTranscripts . . . . .	74
filterUORFs . . . . .	75
fimport . . . . .	75
findFa . . . . .	76
findFromPath . . . . .	77
findLibrariesInFolder . . . . .	77
findMapORFs . . . . .	78
findMaxPeaks . . . . .	79
findNewTSS . . . . .	80
findNGSPairs . . . . .	80
findORFs . . . . .	81
findORFsFasta . . . . .	82
findPeaksPerGene . . . . .	83
findUORFs . . . . .	84
firstEndPerGroup . . . . .	86
firstExonPerGroup . . . . .	87

firstStartPerGroup . . . . .	87
floss . . . . .	88
footprints.analysis . . . . .	90
fpkm . . . . .	90
fpkm_calc . . . . .	92
fractionLength . . . . .	92
fread.bed . . . . .	93
gcContent . . . . .	94
getGAlignments . . . . .	95
getGAlignmentsPairs . . . . .	95
getGenomeAndAnnotation . . . . .	96
getGRanges . . . . .	98
getNGenesCoverage . . . . .	99
getWeights . . . . .	99
get_genome_fasta . . . . .	100
get_genome_gtf . . . . .	101
get_noncoding_rna . . . . .	102
get_phix_genome . . . . .	104
get_silva_rRNA . . . . .	105
groupGRangesBy . . . . .	105
groupings . . . . .	106
gSort . . . . .	107
hasHits . . . . .	107
heatMapL . . . . .	108
heatMapRegion . . . . .	109
heatMap_single . . . . .	111
import.bedo . . . . .	112
import.bedoc . . . . .	113
import.ofst . . . . .	114
importGtfFromTxdb . . . . .	115
initiationScore . . . . .	115
insideOutsideORF . . . . .	116
install.fastp . . . . .	118
install.sratoolkit . . . . .	119
is.grl . . . . .	119
is.gr_or_grl . . . . .	120
is.ORF . . . . .	120
is.range . . . . .	121
isInFrame . . . . .	121
isOverlapping . . . . .	122
isPeriodic . . . . .	123
kozakHeatmap . . . . .	123
kozakSequenceScore . . . . .	125
kozak_IR_ranking . . . . .	126
lastExonEndPerGroup . . . . .	127
lastExonPerGroup . . . . .	128
lastExonStartPerGroup . . . . .	128
libNames . . . . .	129
libraryTypes . . . . .	129
list.experiments . . . . .	130
loadRegion . . . . .	131
loadRegions . . . . .	131

loadTranscriptType . . . . .	132
loadTxdb . . . . .	133
longestORFs . . . . .	134
mainNames . . . . .	134
makeExonRanks . . . . .	135
makeORFNames . . . . .	135
makeSummarizedExperimentFromBam . . . . .	136
mapToGRanges . . . . .	137
matchColors . . . . .	138
matchNaming . . . . .	138
matchSeqStyle . . . . .	139
mergeFastq . . . . .	139
metaWindow . . . . .	140
nrow,experiment-method . . . . .	142
numCodons . . . . .	142
numExonsPerGroup . . . . .	143
optimizeReads . . . . .	143
orfID . . . . .	144
ORFik.template.experiment . . . . .	144
ORFikQC . . . . .	145
orfScore . . . . .	146
organism.df . . . . .	147
outputLibs . . . . .	148
pasteDir . . . . .	149
percentage_to_ratio . . . . .	150
plotHelper . . . . .	150
pmapFromTranscriptF . . . . .	151
pmapToTranscriptF . . . . .	152
prettyScoring . . . . .	153
pseudo.transform . . . . .	153
pSitePlot . . . . .	154
QCplots . . . . .	155
QCreport . . . . .	156
QCstats . . . . .	157
QCstats.plot . . . . .	157
QC_count_tables . . . . .	158
rankOrder . . . . .	159
read.experiment . . . . .	160
readBam . . . . .	161
readLengthTable . . . . .	162
readWidths . . . . .	162
readWig . . . . .	163
reassignTSSbyCage . . . . .	164
reassignTxDbByCage . . . . .	166
reduceKeepAttr . . . . .	167
regionPerReadLength . . . . .	168
remakeTxdbExonIds . . . . .	170
remove.experiments . . . . .	170
remove.file_ext . . . . .	171
removeMetaCols . . . . .	171
removeORFsWithinCDS . . . . .	172
removeORFsWithSameStartAsCDS . . . . .	172

removeORFsWithSameStopAsCDS	173
removeORFsWithStartInsideCDS	173
removeTxdbExons	174
removeTxdbTranscripts	174
rename.SRA.files	175
repNames	175
restrictTSSByUpstreamLeader	176
reverseMinusStrandPerGroup	176
RiboQC.plot	177
ribosomeReleaseScore	177
ribosomeStallingScore	179
rnaNormalize	180
save.experiment	180
savePlot	181
scaledWindowPositions	182
scoreSummarizedExperiment	183
seqnamesPerGroup	184
shiftFootprints	184
shiftFootprintsByExperiment	186
shiftPlots	188
shifts.load	189
show.experiment-method	189
simpleLibs	190
sortPerGroup	191
splitIn3Tx	192
stageNames	193
STAR.align.folder	193
STAR.align.single	197
STAR.allsteps.multiQC	199
STAR.index	200
STAR.install	201
STAR.multiQC	202
STAR.remove.crashed.genome	203
startCodons	203
startDefinition	204
startRegion	205
startRegionCoverage	206
startRegionString	207
startSites	207
stopCodons	208
stopDefinition	209
stopRegion	210
stopSites	211
strandBool	211
strandPerGroup	212
subsetCoverage	213
subsetToFrame	213
te.plot	214
te.table	215
te_rna.plot	216
tile1	217
tissueNames	218

TOP.Motif.ecdf . . . . .	218
topMotif . . . . .	220
transcriptWindow . . . . .	221
transcriptWindow1 . . . . .	222
transcriptWindowPer . . . . .	223
translationalEff . . . . .	224
trimming.table . . . . .	226
trim_detection . . . . .	227
txNames . . . . .	227
txNamesToGeneNames . . . . .	228
txSeqsFromFa . . . . .	229
uniqueGroups . . . . .	229
uniqueOrder . . . . .	230
unlistGrl . . . . .	231
uORFSearchSpace . . . . .	232
updateTxdbRanks . . . . .	233
updateTxdbStartSites . . . . .	234
upstreamFromPerGroup . . . . .	234
upstreamOfPerGroup . . . . .	235
validateExperiments . . . . .	235
validGRL . . . . .	236
validSeqlevels . . . . .	236
widthPerGroup . . . . .	237
windowCoveragePlot . . . . .	238
windowPerGroup . . . . .	239
windowPerReadLength . . . . .	240
windowPerTranscript . . . . .	242
xAxisScaler . . . . .	243
yAxisScaler . . . . .	243

<b>Index</b>	<b>244</b>
--------------	------------

---

ORFik-package

*ORFik for analysis of open reading frames.*

---

## Description

Main goals:

1. Finding Open Reading Frames (very fast) in the genome of interest or on the set of transcripts/sequences.
2. Utilities for metaplots of RiboSeq coverage over gene START and STOP codons allowing to spot the shift.
3. Shifting functions for the RiboSeq data.
4. Finding new Transcription Start Sites with the use of CageSeq data.
5. Various measurements of gene identity e.g. FLOSS, coverage, ORFscore, entropy that are recreated based on many scientific publications.
6. Utility functions to extend GenomicRanges for faster grouping, splitting, tiling etc.

**Author(s)**

**Maintainer:** Haakon Tjeldnes <hauken\_heyken@hotmail.com> [data contributor]

Authors:

- Kornel Labun <kornellabun@gmail.com> [copyright holder]

Other contributors:

- Katarzyna Chyzynska <katchyz@gmail.com> [contributor, data contributor]
- Yamila Torres Cleuren <yamilatortrescleuren@gmail.com> [contributor, thesis advisor]
- Evind Valen <eivind.valen@gmail.com> [thesis advisor, funder]

**See Also**

Useful links:

- <https://github.com/Roleren/ORFik>
- Report bugs at <https://github.com/Roleren/ORFik/issues>

---

addCdsOnLeaderEnds      *Extends leaders downstream*

---

**Description**

When finding uORFs, often you want to allow them to end inside the cds.

**Usage**

```
addCdsOnLeaderEnds(fiveUTRs, cds, onlyFirstExon = FALSE)
```

**Arguments**

fiveUTRs	The 5' leader sequences as GRangesList
cds	If you want to extend 5' leaders downstream, to catch uorfs going into cds, include it.
onlyFirstExon	logical (F), include whole cds or only first exons.

**Details**

This is a simple way to do that

**Value**

a GRangesList of cds exons added to ends

**See Also**

Other uorfs: [filterUORFs\(\)](#), [removeORFsWithSameStartAsCDS\(\)](#), [removeORFsWithSameStopAsCDS\(\)](#), [removeORFsWithStartInsideCDS\(\)](#), [removeORFsWithinCDS\(\)](#), [uORFSearchSpace\(\)](#)



---

addNewTSSOnLeaders	<i>Add cage max peaks as new transcript start sites for each 5' leader (*) strands are not supported, since direction must be known.</i>
--------------------	--

---

### Description

Add cage max peaks as new transcript start sites for each 5' leader (\*) strands are not supported, since direction must be known.

### Usage

```
addNewTSSOnLeaders(fiveUTRs, maxPeakPosition, removeUnused, cageMcol)
```

### Arguments

fiveUTRs	(GRangesList) The 5' leaders or full transcript sequences
maxPeakPosition	The max peak for each 5' leader found by cage
removeUnused	logical (FALSE), if False: (standard is to set them to original annotation), If TRUE: remove leaders that did not have any cage support.
cageMcol	a logical (FALSE), if TRUE, add a meta column to the returned object with the raw CAGE counts in support for new TSS.

### Value

a GRanges object of first exons

---

allFeaturesHelper	<i>Calculate the features in computeFeatures function</i>
-------------------	---

---

### Description

Not used directly, calculates all features internally for computeFeatures.

### Usage

```
allFeaturesHelper(
  grl,
  RFP,
  RNA,
  tx,
  fiveUTRs,
  cds,
  threeUTRs,
  faFile,
  riboStart,
  riboStop,
  sequenceFeatures,
  grl.is.sorted,
```

```

weight.RFP = 1L,
weight.RNA = 1L,
st = NULL
)

```

### Arguments

<code>grl</code>	a <a href="#">GRangesList</a> object with usually ORFs, but can also be either leaders, cds', 3' utrs, etc. This is the regions you want to score.
<code>RFP</code>	RiboSeq reads as <a href="#">GAlignments</a> , <a href="#">GRanges</a> or <a href="#">GRangesList</a> object
<code>RNA</code>	RnaSeq reads as <a href="#">GAlignments</a> , <a href="#">GRanges</a> or <a href="#">GRangesList</a> object
<code>tx</code>	a <a href="#">GrangesList</a> of transcripts, normally called from: <code>exonsBy(Gtf, by = "tx", use.names = T)</code> only add this if you are not including Gtf file If you are using CAGE, you do not need to reassign these to the cage peaks, it will do it for you.
<code>fiveUTRs</code>	<code>fiveUTRs</code> as <a href="#">GRangesList</a> , if you used cage-data to extend 5' utrs, remember to input CAGE assigned version and not original!
<code>cds</code>	a <a href="#">GRangesList</a> of coding sequences
<code>threeUTRs</code>	a <a href="#">GrangesList</a> of transcript 3' utrs, normally called from: <code>threeUTRsByTranscript(Gtf, use.names = T)</code>
<code>faFile</code>	a path to fasta indexed genome, an open <a href="#">FaFile</a> , a <a href="#">BSgenome</a> , or path to <a href="#">ORFik experiment</a> with valid genome.
<code>riboStart</code>	usually 26, the start of the floss interval, see <code>?floss</code>
<code>riboStop</code>	usually 34, the end of the floss interval
<code>sequenceFeatures</code>	a logical, default TRUE, include all sequence features, that is: <code>Kozak</code> , <code>fractionLengths</code> , <code>distORFCDS</code> , <code>isInFrame</code> , <code>isOverlapping</code> and <code>rankInTx</code>
<code>grl.is.sorted</code>	logical (F), a speed up if you know argument <code>grl</code> is sorted, set this to TRUE.
<code>weight.RFP</code>	a vector (default: 1L). Can also be character name of column in RFP. As in <code>translationalEff(weight = "score")</code> for: <code>GRanges("chr1", 1, "+", score = 5)</code> , would mean score column tells that this alignment region was found 5 times.
<code>weight.RNA</code>	Same as <code>weightRFP</code> but for RNA weights. (default: 1L)
<code>st</code>	(NULL), if defined must be: <code>st = startRegion(grl, tx, T, -3, 9)</code>

### Value

a `data.table` with features

---

artificial.orfs

*Create small artificial orfs from cds*

---

### Description

Usefull to see if short ORFs prediction is dependent on length.

Split cds first in two, a start part and stop part. Then say how large the two parts can be and merge them together. It will sample a value in range give.

Parts will be forced to not overlap and can not extend outside original cds

**Usage**

```
artificial.orfs(
  cds,
  start5 = 1,
  end5 = 4,
  start3 = -4,
  end3 = 0,
  bin.if.few = TRUE
)
```

**Arguments**

cds	a GRangesList of orfs, must have width $\% \% 3 == 0$ and length $\geq 6$
start5	integer, default: 1 (start of orf)
end5	integer, default: 4 (max 4 codons from start codon)
start3	integer, default -4 (max 4 codons from stop codon)
end3	integer, default: 0 (end of orf)
bin.if.few	logical, default TRUE, instead of per codon, do per 2, 3, 4 codons if you have few samples compared to lengths wanted, If you have 4 cds' and you want 7 different lengths, which is the standard, it will give you possible nt length: 6-12-18-24 instead of original 6-9-12-15-18-21-24. If you have more than 30x cds than lengths wanted this is skipped. (for default arguments this is: $7*30 = 210$ cds)

**Details**

If artificial cds length is not divisible by 2, like 3 codons, the second codon will always be from the start region etc.

Also If there are many very short original cds, the distribution will be skewed towards more smaller artificial cds.

**Value**

GRangesList of new ORFs (sorted: + strand increasing start, - strand decreasing start)

---

assignAnnotations      *Overlaps GRanges object with provided annotations.*

---

**Description**

It will return same list of GRanges, but with metadata columns: transcript\_id - id of transcripts that overlap with each ORF gene\_id - id of gene that this transcript belongs to isoform - for coding protein alignment in relation to cds on corresponding transcript, for non-coding transcripts alignment in relation to the transcript.

**Usage**

```
assignAnnotations(ORFs, con)
```

**Arguments**

- ORFs - GRanges or GRangesList object of your ORFs.  
 con - Path to gtf file with annotations.

**Value**

A GRanges object of your ORFs with metadata columns 'gene', 'transcript', 'isoform' and 'biotype'.

---

assignFirstExonsStartSite

*Reassign the start positions of the first exons per group in grl*

---

**Description**

Per group in GRangesList, assign the most upstream site.

**Usage**

```
assignFirstExonsStartSite(grl, newStarts)
```

**Arguments**

- grl a [GRangesList](#) object  
 newStarts an integer vector of same length as grl, with new start values (absolute coordinates, not relative)

**Details**

make sure your grl is sorted, since start of "-" strand objects should be the max end in group, use `ORFik:::sortPerGroup(grl)` to get sorted grl.

**Value**

the same GRangesList with new start sites

**See Also**

Other GRanges: [assignLastExonsStopSite\(\)](#), [downstreamFromPerGroup\(\)](#), [downstreamOfPerGroup\(\)](#), [upstreamFromPerGroup\(\)](#), [upstreamOfPerGroup\(\)](#)

---

 assignLastExonsStopSite

*Reassign the stop positions of the last exons per group*


---

### Description

Per group in GRangesList, assign the most upstream site.

### Usage

```
assignLastExonsStopSite(grl, newStops)
```

### Arguments

grl	a <a href="#">GRangesList</a> object
newStops	an integer vector of same length as grl, with new start values (absolute coordinates, not relative)

### Details

make sure your grl is sorted, since stop of "-" strand objects should be the min start in group, use `ORFik:::sortPerGroup(grl)` to get sorted grl.

### Value

the same GRangesList with new stop sites

### See Also

Other GRanges: [assignFirstExonsStartSite\(\)](#), [downstreamFromPerGroup\(\)](#), [downstreamOfPerGroup\(\)](#), [upstreamFromPerGroup\(\)](#), [upstreamOfPerGroup\(\)](#)

---

 assignTSSByCage

*Input a txdb and add a 5' leader for each transcript, that does not have one.*


---

### Description

For all cds in txdb, that does not have a 5' leader: Start at 1 base upstream of cds and use CAGE, to assign leader start. All these leaders will be 1 exon based, if you really want exon splicings, you can use exon prediction tools, or run sequencing experiments.

**Usage**

```

assignTSSByCage(
  txdb,
  cage,
  extension = 1000,
  filterValue = 1,
  restrictUpstreamToTx = FALSE,
  removeUnused = FALSE,
  preCleanup = TRUE
)

```

**Arguments**

txdb	a TxDb file, a path to one of: (.gtf, .gff, .gff2, .gff2, .db or .sqlite) or an ORFik experiment
cage	Either a filePath for the CageSeq file as .bed, .bam or .wig, with possible compressions (".gzip", ".gz", ".bgz"), or already loaded CageSeq peak data as GRanges or GAlignment. NOTE: If it is a .bam file, it will add a score column by running: <code>convertToOneBasedRanges(cage, method = "5prime", addScoreColumn = TRUE)</code> The score column is then number of replicates of read, if score column is something else, like read length, set the score column to NULL first.
extension	The maximum number of bases upstream of the TSS to search for CageSeq peak.
filterValue	The minimum number of reads on cage position, for it to be counted as possible new tss. (represented in score column in CageSeq data) If you already filtered, set it to 0.
restrictUpstreamToTx	a logical (FALSE). If TRUE: restrict leaders to not extend closer than 5 bases from closest upstream leader, set this to TRUE.
removeUnused	logical (FALSE), if FALSE: (standard is to set them to original annotation), If TRUE: remove leaders that did not have any cage support.
preCleanup	logical (TRUE), if TRUE, remove all reads in region (-5:-1, 1:5) of all original tss in leaders. This is to keep original TSS if it is only +/- 5 bases from the original.

**Details**

Given a TxDb object, reassign the start site per transcript using max peaks from CageSeq data. A max peak is defined as new TSS if it is within boundary of 5' leader range, specified by 'extension' in bp. A max peak must also be higher than minimum CageSeq peak cutoff specified in 'filterValue'. The new TSS will then be positioned where the cage read (with highest read count in the interval).

**Value**

a TxDb object of reassigned transcripts

**See Also**

Other CAGE: [reassignTSSbyCage\(\)](#), [reassignTxDbByCage\(\)](#)

**Examples**

```
txdbFile <- system.file("extdata", "hg19_knownGene_sample.sqlite",
  package = "GenomicFeatures")
cagePath <- system.file("extdata", "cage-seq-heart.bed.bgz",
  package = "ORFik")

## Not run:
  assignTSSByCage(txdbFile, cagePath)
  Minimum 20 cage tags for new TSS
  assignTSSByCage(txdbFile, cagePath, filterValue = 20)

## End(Not run)
```

asTX

*Map genomic to transcript coordinates by reference***Description**

Map range coordinates between features in the genome and transcriptome (reference) space.

**Usage**

```
asTX(
  grl,
  reference,
  ignore.strand = FALSE,
  x.is.sorted = TRUE,
  tx.is.sorted = TRUE
)
```

**Arguments**

<code>grl</code>	a <a href="#">GRangesList</a> of ranges within the reference, <code>grl</code> must have column called names that gives grouping for result
<code>reference</code>	a <a href="#">GRangesList</a> of ranges that include and are bigger or equal to <code>grl</code> ig. <code>cds</code> is <code>grl</code> and <code>gene</code> can be <code>reference</code>
<code>ignore.strand</code>	When <code>ignore.strand</code> is <code>TRUE</code> , <code>strand</code> is ignored in overlaps operations (i.e., all strands are considered "+") and the strand in the output is '*'. When <code>ignore.strand</code> is <code>FALSE</code> (default) <code>strand</code> in the output is taken from the transcripts argument. When transcripts is a <a href="#">GRangesList</a> , all inner list elements of a common list element must have the same strand or an error is thrown. Mapped position is computed by counting from the transcription start site (TSS) and is not affected by the value of <code>ignore.strand</code> .
<code>x.is.sorted</code>	if <code>x</code> is a <a href="#">GRangesList</a> object, are "-" strand groups pre-sorted in decreasing order within group, default: <code>TRUE</code>
<code>tx.is.sorted</code>	if transcripts is a <a href="#">GRangesList</a> object, are "-" strand groups pre-sorted in decreasing order within group, default: <code>TRUE</code>

**Details**

Similar to `GenomicFeatures`' `pmapToTranscripts`, but in this version the `grl` ranges are compared to reference ranges with same name, not by index. And it has a security fix.

**Value**

a GRangesList in transcript coordinates

**See Also**

Other ExtendGenomicRanges: [coveragePerTiling\(\)](#), [extendLeaders\(\)](#), [extendTrailers\(\)](#), [reduceKeepAttr\(\)](#), [tile1\(\)](#), [txSeqsFromFa\(\)](#), [windowPerGroup\(\)](#)

---

bamVarName

*Get library variable names from ORFik [experiment](#)*

---

**Description**

What will each sample be called given the columns of the experiment?

**Usage**

```
bamVarName(
  df,
  skip.replicate = length(unique(df$rep)) == 1,
  skip.condition = length(unique(df$condition)) == 1,
  skip.stage = length(unique(df$stage)) == 1,
  skip.fraction = length(unique(df$fraction)) == 1,
  skip.experiment = !df@expInVarName,
  skip.libtype = FALSE
)
```

**Arguments**

`df` an ORFik [experiment](#)

`skip.replicate` a logical (FALSE), don't include replicate in variable name.

`skip.condition` a logical (FALSE), don't include condition in variable name.

`skip.stage` a logical (FALSE), don't include stage in variable name.

`skip.fraction` a logical (FALSE), don't include fraction

`skip.experiment` a logical (FALSE), don't include experiment

`skip.libtype` a logical (FALSE), don't include libtype

**Value**

variable names of libraries (character vector)

**See Also**

Other ORFik\_experiment: [ORFik.template.experiment\(\)](#), [create.experiment\(\)](#), [experiment-class](#), [filepath\(\)](#), [libraryTypes\(\)](#), [organism.df\(\)](#), [outputLibs\(\)](#), [read.experiment\(\)](#), [save.experiment\(\)](#), [validateExperiments\(\)](#)



## Examples

```
df <- ORFik.template.experiment()
bamVarName(df)

## without libtype
bamVarName(df, skip.libtype = TRUE)
## Without experiment name
bamVarName(df, skip.experiment = TRUE)
```

---

bamVarNamePicker	<i>Get variable name per filepath in experiment</i>
------------------	---

---

## Description

Get variable name per filepath in experiment

## Usage

```
bamVarNamePicker(
  df,
  skip.replicate = FALSE,
  skip.condition = FALSE,
  skip.stage = FALSE,
  skip.fraction = FALSE,
  skip.experiment = FALSE,
  skip.libtype = FALSE
)
```

## Arguments

df                    an ORFik [experiment](#)

skip.replicate    a logical (FALSE), don't include replicate in variable name.

skip.condition    a logical (FALSE), don't include condition in variable name.

skip.stage        a logical (FALSE), don't include stage in variable name.

skip.fraction     a logical (FALSE), don't include fraction

skip.experiment                    a logical (FALSE), don't include experiment

skip.libtype      a logical (FALSE), don't include libtype

## Value

variable name of library (character vector)

---

bedToGR	<i>Converts bed style data.frame to GRanges</i>
---------	---

---

**Description**

For info on columns, see: <https://www.ensembl.org/info/website/upload/bed.html>

**Usage**

```
bedToGR(x, skip.name = TRUE)
```

**Arguments**

x	A <a href="#">data.frame</a> from imported bed-file, to convert to GRanges
skip.name	default (TRUE), skip name column (column 4)

**Value**

a [GRanges](#) object from bed

**See Also**

Other utils: [convertToOneBasedRanges\(\)](#), [export.bed12\(\)](#), [export.wiggle\(\)](#), [fimport\(\)](#), [findFa\(\)](#), [fread.bed\(\)](#), [optimizeReads\(\)](#), [readBam\(\)](#), [readWig\(\)](#)

---

cellLineNames	<i>Get cell-line name variants</i>
---------------	------------------------------------

---

**Description**

Used to standardize nomenclature for experiments.

Example: THP-1 is main naming, but a variant is THP1 THP1 will then be renamed to THP-1

**Usage**

```
cellLineNames()
```

**Value**

a [data.table](#) with 2 columns, the main name, and all name variants of the main name in second column as a list.

**See Also**

Other experiment\_naming: [conditionNames\(\)](#), [libNames\(\)](#), [mainNames\(\)](#), [repNames\(\)](#), [stageNames\(\)](#), [tissueNames\(\)](#)

---

changePointAnalysis    *Get the offset for specific RiboSeq read width*

---

### Description

Creates sliding windows of transcript normalized counts per position and check which window has most in upstream window vs downstream window. Pick the position with highest absolute value maximum of the window difference. Checks windows with split sites between positions -17 to -7, where 0 is TIS. Normally you expect the shift around -12.

### Usage

```
changePointAnalysis(  
  x,  
  feature = "start",  
  max.pos = 40L,  
  interval = seq.int(14L, 24L)  
)
```

### Arguments

x	a vector with count per position to analyse, assumes the zero position (TIS) is in the middle + 1 (position 0). Default it is size 60, from -30 to 29 in p-shifting
feature	(character) either "start" or "stop"
max.pos	integer, default 40L, subset x to go from index 1 to max.pos, if tail is not relevant.
interval	integer vector , default seq.int(14L, 24L). Separation points for upstream and downstream windows. That is (+/- 5 from -12) position.

### Details

Transcript normalized means per CDS TIS region, count reads per position, divide that number per position by the total of that transcript, then sum up these numbers per position for all transcripts.

### Value

a single numeric offset, -12 would mean p-site is 12 bases upstream

### See Also

Other pshifting: [detectRibosomeShifts\(\)](#), [shiftFootprintsByExperiment\(\)](#), [shiftFootprints\(\)](#)

---

`checkRFP`*Helper Function to check valid RFP input*

---

**Description**

Helper Function to check valid RFP input

**Usage**

```
checkRFP(class)
```

**Arguments**

`class`, the given class of RFP object

**Value**

NULL, stop if invalid object

**See Also**

Other validity: [checkRNA\(\)](#), [is.ORF\(\)](#), [is.gr\\_or\\_grl\(\)](#), [is.grl\(\)](#), [is.range\(\)](#), [validGRL\(\)](#), [validSeqlevels\(\)](#)

---

`checkRNA`*Helper Function to check valid RNA input*

---

**Description**

Helper Function to check valid RNA input

**Usage**

```
checkRNA(class)
```

**Arguments**

`class`, the given class of RNA object

**Value**

NULL, stop if unvalid object

**See Also**

Other validity: [checkRFP\(\)](#), [is.ORF\(\)](#), [is.gr\\_or\\_grl\(\)](#), [is.grl\(\)](#), [is.range\(\)](#), [validGRL\(\)](#), [validSeqlevels\(\)](#)

---

codonSumsPerGroup      *Get read hits per codon*

---

### Description

Helper for entropy function, normally not used directly. Separate each group into tuples (abstract codons). Gives sum for each tuple within each group.

### Usage

```
codonSumsPerGroup(gr1, reads, weight = "score", is.sorted = FALSE)
```

### Arguments

gr1	GRangesList or GRanges of your ranges
reads	GRanges object of your reads.
weight	(default: 'score'), if defined a character name of valid meta column in subject. GRanges("chr1", 1, "+", score = 5), would mean score column tells that this alignment region was found 5 times. ORFik .bedo files, contains a score column like this. As do CAGEr CAGE files and many other package formats. You can also assign a score column manually.
is.sorted	logical (FALSE), is gr1 sorted. That is + strand groups in increasing ranges (1,2,3), and - strand groups in decreasing ranges (3,2,1)

### Details

Example: counts c(1,0,0,1), with reg\_len = 2, gives c(1,0) and c(0,1), these are summed and returned as data.table 10 bases, will give 3 codons, 1 base codons does not exist.

### Value

a data.table with codon sums

---

collapse.by.scores      *Merge reads by sum of existing scores*

---

### Description

If you have multiple reads at a same location but different read lengths, specified in meta column "size", it will sum up the scores (number of replicates) for all reads at that position.

### Usage

```
collapse.by.scores(x)
```

### Arguments

x	a GRanges object
---	------------------

**Value**

merged GRanges object

**Examples**

```
gr_s1 <- rep(GRanges("chr1", 1:10, "+"), 2)
gr_s2 <- GRanges("chr1", 1:12, "+")
gr2 <- GRanges("chr1", 21:40, "+")
gr <- c(gr_s1, gr_s2, gr2)
res <- convertToOneBasedRanges(gr,
  addScoreColumn = TRUE, addSizeColumn = TRUE)
ORFik::collapse.by.scores(res)
```

---

collapse.fastq

*Very fast fastq/fastq collapse*


---

**Description**

For each unique read in the file, collapse into 1 and state in the fasta header how many reads existed of that type. This is done after trimming usually, works best for reads < 50 read length. Not so effective for 150 bp length mRNA-seq etc.

**Usage**

```
collapse.fastq(
  files,
  outdir = file.path(dirname(files[1]), "collapsed"),
  header.out.format = "ribotoolkit",
  compress = FALSE
)
```

**Arguments**

files	paths to fasta / fastq files to collapse.
outdir	outdir to save files, default: file.path(dirname(files[1]), "collapsed"). Inside same folder as input files, then create subfolder "collapsed", and add a prefix of "collapsed_" to the output names in that folder.
header.out.format	character, default "ribotoolkit", else must be "fastx". How the read header of the output fasta should be formatted: ribotoolkit: "<seq1_x55", sequence 1 has 55 duplicated reads collapsed. fastx: "<1-55", sequence 1 has 55 duplicated reads collapsed
compress	logical, default FALSE

**Value**

invisible(NULL)

## Examples

```
fastq.folder <- tempdir() # <- Your fastq files
infile <- dir(fastq.folder, "*.fastq", full.names = TRUE)
# collapse.fastq(infile)
```

---

```
collapseDuplicatedReads
      Collapse duplicated reads
```

---

## Description

For every GRanges, GAlignments read, with the same: seqname, start, (cigar) / width and strand, collapse and give a new meta column called "score", which contains the number of duplicates of that read. If score column already exists, will return input object!

## Usage

```
collapseDuplicatedReads(x, ...)
```

## Arguments

x	a GRanges, GAlignments or GAlignmentPairs object
...	alternative arguments. addScoreColumn = TRUE, if FALSE, only collapse and not add score column.

## Value

a GRanges, GAlignments or GAlignmentPairs object, same as input

## Examples

```
gr <- rep(GRanges("chr1", 1:10,"+"), 2)
collapseDuplicatedReads(gr)
```

---

```
collapseDuplicatedReads,GAlignmentPairs-method
      Collapse duplicated reads
```

---

## Description

For every GRanges, GAlignments read, with the same: seqname, start, (cigar) / width and strand, collapse and give a new meta column called "score", which contains the number of duplicates of that read. If score column already exists, will return input object!

## Usage

```
## S4 method for signature 'GAlignmentPairs'
collapseDuplicatedReads(x, addScoreColumn = TRUE)
```

**Arguments**

x                    a GRanges, GAlignments or GAlignmentPairs object  
addScoreColumn = TRUE, if FALSE, only collapse and not add score column.

**Value**

a GRanges, GAlignments or GAlignmentPairs object, same as input

**Examples**

```
gr <- rep(GRanges("chr1", 1:10,"+"), 2)
collapseDuplicatedReads(gr)
```

---

*collapseDuplicatedReads,GAlignments-method*  
*Collapse duplicated reads*

---

**Description**

For every GRanges, GAlignments read, with the same: seqname, start, (cigar) / width and strand, collapse and give a new meta column called "score", which contains the number of duplicates of that read. If score column already exists, will return input object!

**Usage**

```
## S4 method for signature 'GAlignments'
collapseDuplicatedReads(x, addScoreColumn = TRUE)
```

**Arguments**

x                    a GRanges, GAlignments or GAlignmentPairs object  
addScoreColumn = TRUE, if FALSE, only collapse and not add score column.

**Value**

a GRanges, GAlignments or GAlignmentPairs object, same as input

**Examples**

```
gr <- rep(GRanges("chr1", 1:10,"+"), 2)
collapseDuplicatedReads(gr)
```



---

collapseDuplicatedReads, GRanges-method  
*Collapse duplicated reads*

---

## Description

For every GRanges, GAlignments read, with the same: seqname, start, (cigar) / width and strand, collapse and give a new meta column called "score", which contains the number of duplicates of that read. If score column already exists, will return input object!

## Usage

```
## S4 method for signature 'GRanges'  
collapseDuplicatedReads(  
  x,  
  addScoreColumn = TRUE,  
  addSizeColumn = FALSE,  
  reuse.score.column = TRUE  
)
```

## Arguments

`x` a GRanges, GAlignments or GAlignmentPairs object

`addScoreColumn = TRUE`, if FALSE, only collapse and not keep score column.

`addSizeColumn` logical (FALSE), if TRUE, add a size column that for each read, that gives original width of read. Useful if you need original read lengths. This takes care of soft clips etc. If collapsing reads, each unique range will be grouped also by size.

`reuse.score.column` logical (TRUE), if addScoreColumn is TRUE, and a score column exists, will sum up the scores to create a new score. If FALSE, will skip old score column and create new according to number of replicated reads after conversion. If addScoreColumn is FALSE, this argument is ignored.

## Value

a GRanges, GAlignments or GAlignmentPairs object, same as input

## Examples

```
gr <- rep(GRanges("chr1", 1:10, "+"), 2)  
collapseDuplicatedReads(gr)
```

---

combn.pairs	<i>Create all unique combinations pairs possible</i>
-------------	--

---

**Description**

Given a character vector, get all unique combinations of 2.

**Usage**

```
combn.pairs(x)
```

**Arguments**

x                    a character vector, will unique elements for you.

**Value**

a list of character vector pairs

---

computeFeatures	<i>Get all possible features in ORFik</i>
-----------------	---

---

**Description**

If you want to get all the NGS and/or sequence features easily, you can use this function. Each feature have a link to an article describing its creation and idea behind it. Look at the functions in the feature family to see all of them. Example, if you want to know what the "te" column is, check out: ?translationalEff.

If you used CageSeq to reannotate your leaders, your txDB object must contain the reassigned leaders. Use [reassignTxDbByCage()] to get the txdb.

**Usage**

```
computeFeatures(
  grl,
  RFP,
  RNA = NULL,
  Gtf,
  faFile = NULL,
  riboStart = 26,
  riboStop = 34,
  sequenceFeatures = TRUE,
  grl.is.sorted = FALSE,
  weight.RFP = 1L,
  weight.RNA = 1L
)
```

**Arguments**

grl	a <a href="#">GRangesList</a> object with usually ORFs, but can also be either leaders, cds', 3' utrs, etc. This is the regions you want to score.
RFP	RiboSeq reads as <a href="#">GAlignments</a> , <a href="#">GRanges</a> or <a href="#">GRangesList</a> object
RNA	RnaSeq reads as <a href="#">GAlignments</a> , <a href="#">GRanges</a> or <a href="#">GRangesList</a> object
Gtf	a TxDb object of a gtf file or path to gtf, gff .sqlite etc.
faFile	a path to fasta indexed genome, an open <a href="#">FaFile</a> , a BSgenome, or path to <a href="#">ORFik experiment</a> with valid genome.
riboStart	usually 26, the start of the floss interval, see ?floss
riboStop	usually 34, the end of the floss interval
sequenceFeatures	a logical, default TRUE, include all sequence features, that is: Kozak, fractionLengths, distORFCDS, isInFrame, isOverlapping and rankInTx
grl.is.sorted	logical (F), a speed up if you know argument grl is sorted, set this to TRUE.
weight.RFP	a vector (default: 1L). Can also be character name of column in RFP. As in translationalEff(weight = "score") for: GRanges("chr1", 1, "+", score = 5), would mean score column tells that this alignment region was found 5 times.
weight.RNA	Same as weightRFP but for RNA weights. (default: 1L)

**Details**

As a note the library is reduced to only reads overlapping 'tx', so the library size in fpkm calculation is done on this subset. This will help remove rRNA and other contaminants.

Also if you have only unique reads with a weight column, explaining the number of duplicated reads, set weights to make calculations correct. See [getWeights](#)

**Value**

a data.table with scores, each column is one score type, name of columns are the names of the scores, i.g [floss()] or [fpkm()]

**See Also**

Other features: [computeFeaturesCage\(\)](#), [countOverlapsW\(\)](#), [disengagementScore\(\)](#), [distToCds\(\)](#), [distToTSS\(\)](#), [entropy\(\)](#), [floss\(\)](#), [fpkm\\_calc\(\)](#), [fpkm\(\)](#), [fractionLength\(\)](#), [initiationScore\(\)](#), [insideOutsideORF\(\)](#), [isInFrame\(\)](#), [isOverlapping\(\)](#), [kozakSequenceScore\(\)](#), [orfScore\(\)](#), [rankOrder\(\)](#), [ribosomeReleaseScore\(\)](#), [ribosomeStallingScore\(\)](#), [startRegionCoverage\(\)](#), [startRegion\(\)](#), [stopRegion\(\)](#), [subsetCoverage\(\)](#), [translationalEff\(\)](#)

**Examples**

```
# Here we make an example from scratch
# Usually the ORFs are found in orfik, which makes names for you etc.
gtf <- system.file("extdata", "annotations.gtf",
package = "ORFik") ## location of the gtf file
suppressWarnings(txdb <-
  GenomicFeatures::makeTxDbFromGFF(gtf, format = "gtf"))
# use cds' as ORFs for this example
ORFs <- GenomicFeatures::cdsBy(txdb, by = "tx", use.names = TRUE)
ORFs <- makeORFNames(ORFs) # need ORF names
# make Ribo-seq data,
```

```
RFP <- unlistGr1(firstExonPerGroup(ORFs))
suppressWarnings(computeFeatures(ORFs, RFP, Gtf = txdb))
# For more details see vignettes.
```

---

computeFeaturesCage    *Get all possible features in ORFik*

---

## Description

If you have a txdb with correctly reassigned transcripts, use: [computeFeatures()]

## Usage

```
computeFeaturesCage(
  gr1,
  RFP,
  RNA = NULL,
  Gtf = NULL,
  tx = NULL,
  fiveUTRs = NULL,
  cds = NULL,
  threeUTRs = NULL,
  faFile = NULL,
  riboStart = 26,
  riboStop = 34,
  sequenceFeatures = TRUE,
  gr1.is.sorted = FALSE,
  weight.RFP = 1L,
  weight.RNA = 1L
)
```

## Arguments

gr1	a <a href="#">GRangesList</a> object with usually ORFs, but can also be either leaders, cds', 3' utrs, etc. This is the regions you want to score.
RFP	RiboSeq reads as <a href="#">GAlignments</a> , <a href="#">GRanges</a> or <a href="#">GRangesList</a> object
RNA	RnaSeq reads as <a href="#">GAlignments</a> , <a href="#">GRanges</a> or <a href="#">GRangesList</a> object
Gtf	a TxDb object of a gtf file or path to gtf, gff .sqlite etc.
tx	a GrangesList of transcripts, normally called from: exonsBy(Gtf, by = "tx", use.names = T) only add this if you are not including Gtf file If you are using CAGE, you do not need to reassign these to the cage peaks, it will do it for you.
fiveUTRs	fiveUTRs as GRangesList, if you used cage-data to extend 5' utrs, remember to input CAGE assigned version and not original!
cds	a GRangesList of coding sequences
threeUTRs	a GrangesList of transcript 3' utrs, normally called from: threeUTRsByTranscript(Gtf, use.names = T)
faFile	a path to fasta indexed genome, an open <a href="#">FaFile</a> , a BSgenome, or path to ORFik <a href="#">experiment</a> with valid genome.

riboStart	usually 26, the start of the floss interval, see ?floss
riboStop	usually 34, the end of the floss interval
sequenceFeatures	a logical, default TRUE, include all sequence features, that is: Kozak, fractionLengths, distORFCDS, isInFrame, isOverlapping and rankInTx
grl.is.sorted	logical (F), a speed up if you know argument grl is sorted, set this to TRUE.
weight.RFP	a vector (default: 1L). Can also be character name of column in RFP. As in translationalEff(weight = "score") for: GRanges("chr1", 1, "+", score = 5), would mean score column tells that this alignment region was found 5 times.
weight.RNA	Same as weightRFP but for RNA weights. (default: 1L)

### Details

A specialized version if you don't have a correct txdb, for example with CAGE reassigned leaders while txdb is not updated. It is 2x faster for tested data. The point of this function is to give you the ability to input transcript etc directly into the function, and not load them from txdb. Each feature have a link to an article describing feature, try ?floss

### Value

a data.table with scores, each column is one score type, name of columns are the names of the scores, i.g [floss()] or [fpkm()]

### See Also

Other features: [computeFeatures\(\)](#), [countOverlapsW\(\)](#), [disengagementScore\(\)](#), [distToCds\(\)](#), [distToTSS\(\)](#), [entropy\(\)](#), [floss\(\)](#), [fpkm\\_calc\(\)](#), [fpkm\(\)](#), [fractionLength\(\)](#), [initiationScore\(\)](#), [insideOutsideORF\(\)](#), [isInFrame\(\)](#), [isOverlapping\(\)](#), [kozakSequenceScore\(\)](#), [orfScore\(\)](#), [rankOrder\(\)](#), [ribosomeReleaseScore\(\)](#), [ribosomeStallingScore\(\)](#), [startRegionCoverage\(\)](#), [startRegion\(\)](#), [stopRegion\(\)](#), [subsetCoverage\(\)](#), [translationalEff\(\)](#)

### Examples

```
# a small example without cage-seq data:
# we will find ORFs in the 5' utrs
# and then calculate features on them

if (requireNamespace("BSgenome.Hsapiens.UCSC.hg19")) {
  library(GenomicFeatures)
  # Get the gtf txdb file
  txdbFile <- system.file("extdata", "hg19_knownGene_sample.sqlite",
    package = "GenomicFeatures")
  txdb <- loadDb(txdbFile)

  # Extract sequences of fiveUTRs.
  fiveUTRs <- fiveUTRsByTranscript(txdb, use.names = TRUE)[1:10]
  faFile <- BSgenome.Hsapiens.UCSC.hg19::Hsapiens
  # need to suppress warning because of bug in GenomicFeatures, will
  # be fixed soon.
  tx_seqs <- suppressWarnings(extractTranscriptSeqs(faFile, fiveUTRs))

  # Find all ORFs on those transcripts and get their genomic coordinates
  fiveUTR_ORFs <- findMapORFs(fiveUTRs, tx_seqs)
  unlistedORFs <- unlistGr1(fiveUTR_ORFs)
```

```

# group GRanges by ORFs instead of Transcripts
fiveUTR_ORFs <- groupGRangesBy(unlistedORFs, unlistedORFs$names)

# make some toy ribo seq and rna seq data
starts <- unlistGrl(ORFik:::firstExonPerGroup(fiveUTR_ORFs))
RFP <- promoters(starts, upstream = 0, downstream = 1)
score(RFP) <- rep(29, length(RFP)) # the original read widths

# set RNA seq to duplicate transcripts
RNA <- unlistGrl(exonsBy(txdb, by = "tx", use.names = TRUE))

ORFik:::computeFeaturesCage(grl = fiveUTR_ORFs, RFP = RFP,
  RNA = RNA, Gtf = txdb, faFile = faFile)

}
# See vignettes for more examples

```

---

conditionNames	<i>Get condition name variants</i>
----------------	------------------------------------

---

### Description

Used to standardize nomenclature for experiments.

Example: WT is main naming, but a variant is control control will then be renamed to WT

### Usage

```
conditionNames()
```

### Value

a data.table with 2 columns, the main name, and all name variants of the main name in second column as a list.

### See Also

Other experiment\_naming: [cellLineNames\(\)](#), [libNames\(\)](#), [mainNames\(\)](#), [repNames\(\)](#), [stageNames\(\)](#), [tissueNames\(\)](#)

---

convertLibs	<i>Converted format of NGS libraries</i>
-------------	--

---

### Description

Export as either .ofst, .bedo or .bedoc files.

Export files as .bedo files: It is a bed file with 2 score columns. Gives a massive speedup when cigar string and bam flags are not needed.

Export files as .bedoc files: If cigar is needed, gives you replicates and cigar, so a fast way to load a GAlignment object, other bam flags are lost. If type is bedoc addSizeColumn and method will be ignored.

**Usage**

```

convertLibs(
  df,
  out.dir = dirname(df$filepath[1]),
  addScoreColumn = TRUE,
  addSizeColumn = TRUE,
  must.overlap = NULL,
  method = "None",
  type = "ofst",
  reassign.when.saving = FALSE,
  envir = .GlobalEnv
)

```

**Arguments**

df	an ORFik <a href="#">experiment</a>
out.dir	optional output directory, default: <code>dirname(df\$filepath[1])</code> , if it is NULL, it will just reassign R objects to simplified libraries.
addScoreColumn	logical, default TRUE, if FALSE will not add replicate numbers as score column, see <code>ORFik::convertToOneBasedRanges</code> .
addSizeColumn	logical, default TRUE, if FALSE will not add size (width) as size column, see <code>ORFik::convertToOneBasedRanges</code> . Does not apply for <code>.ofst</code> or <code>.bedoc</code> .
must.overlap	default (NULL), else a <code>GRanges / GRangesList</code> object, so only reads that overlap ( <code>must.overlap</code> ) are kept. This is useful when you only need the reads over transcript annotation or subset etc.
method	character, default "None", the method to reduce ranges, for more info see <a href="#">convertToOneBasedRanges</a>
type	a character of format, default "ofst". Alternatives: "ofst", "wig", "bedo" or "bedoc". Which format you want. Will make a folder within <code>out.dir</code> with this name containing the files.
reassign.when.saving	logical, default FALSE. If TRUE, will reassign library to converted form after saving. Ignored when <code>out.dir = NULL</code> .
envir	which environment to save to, default <code>.GlobalEnv</code>

**Details**

See [export.bedo](#) and [export.bedoc](#) for information on file formats

**Value**

NULL (saves files to disc or R `.GlobalEnv`)

**Examples**

```

df <- ORFik.template.experiment()
#convertLibs(df)
# Keep only 5' ends of reads
#convertLibs(df, method = "5prime")

```

---

convertToOneBasedRanges

*Convert a GRanges Object to 1 width reads*

---

## Description

There are 5 ways of doing this

1. Take 5' ends, reduce away rest (5prime)
2. Take 3' ends, reduce away rest (3prime)
3. Tile to 1-mers and include all (tileAll)
4. Take middle point per GRanges (middle)
5. Get original with metacolumns (None)

You can also do multiple at a time, then output is GRangesList, where each list group is the operation (5prime is [1], 3prime is [2] etc)

Many other ways to do this have their own functions, like startSites and stopSites etc. To retain information on original width, set addSizeColumn to TRUE. To compress data, 1 GRanges object per unique read, set addScoreColumn to TRUE. This will give you a score column with how many duplicated reads there were in the specified region.

## Usage

```
convertToOneBasedRanges(
  gr,
  method = "5prime",
  addScoreColumn = FALSE,
  addSizeColumn = FALSE,
  after.softclips = TRUE,
  along.reference = FALSE,
  reuse.score.column = TRUE
)
```

## Arguments

<code>gr</code>	GRanges, GAlignment or GAlignmentPairs object to reduce.
<code>method</code>	the method to reduce ranges, see info. (5prime default)
<code>addScoreColumn</code>	logical (FALSE), if TRUE, add a score column that sums up the hits per unique range. This will make each read unique, so that each read is 1 time, and score column gives the number of collapsed hits. A useful compression. If addSizeColumn is FALSE, it will not differentiate between reads with same start and stop, but different length. If addSizeColumn is FALSE, it will remove it. Collapses after conversion.
<code>addSizeColumn</code>	logical (FALSE), if TRUE, add a size column that for each read, that gives original width of read. Useful if you need original read lengths. This takes care of soft clips etc. If collapsing reads, each unique range will be grouped also by size.
<code>after.softclips</code>	logical (TRUE), include softclips in width. Does not apply if along.reference is TRUE.



**along.reference**

logical (FALSE), example: The cigar "26MI2" is by default width 28, but if along.reference is TRUE, it will be 26. The length of the read along the reference. Also "1D20M" will be 21 if by along.reference is TRUE. Intronic regions (cigar: N) will be removed. So: "1M200N19M" is 20, not 220.

**reuse.score.column**

logical (TRUE), if addScoreColumn is TRUE, and a score column exists, will sum up the scores to create a new score. If FALSE, will skip old score column and create new according to number of replicated reads after conversion. If addScoreColumn is FALSE, this argument is ignored.

**Details**

NOTE: For special case of GAlignmentPairs, 5prime will only use left (first) 5' end and read and 3prime will use only right (last) 3' end of read in pair. tileAll and middle can possibly find point that are not in the reads since: lets say pair is 1-5 and 10-15, middle is 7, which is not in the read.

**Value**

Converted GRanges object

**See Also**

Other utils: [bedToGR\(\)](#), [export.bed12\(\)](#), [export.wiggle\(\)](#), [fimport\(\)](#), [findFa\(\)](#), [fread.bed\(\)](#), [optimizeReads\(\)](#), [readBam\(\)](#), [readWig\(\)](#)

**Examples**

```
gr <- GRanges("chr1", 1:10, "+")
# 5 prime ends
convertToOneBasedRanges(gr)
# is equal to convertToOneBasedRanges(gr, method = "5prime")
# 3 prime ends
convertToOneBasedRanges(gr, method = "3prime")
# With lengths
convertToOneBasedRanges(gr, addSizeColumn = TRUE)
# With score (# of replicates)
gr <- rep(gr, 2)
convertToOneBasedRanges(gr, addSizeColumn = TRUE, addScoreColumn = TRUE)
```

---

correlation.plots

*Correlation plots between all samples*

---

**Description**

Get 2 correlation plots of raw counts and  $\log_2(\text{count} + 1)$  over selected region in: `c("mrna", "leaders", "cds", "trailers")`

**Usage**

```
correlation.plots(
  df,
  output.dir,
  region = "mrna",
  type = "fpkm",
  height = 400,
  width = 400,
  size = 0.15
)
```

**Arguments**

df	an ORFik <a href="#">experiment</a>
output.dir	directory to save to, 2 files named: cor_plot.png and cor_plot_log2.png
region	a character (default: mrna), make raw count matrices of whole mrnas or one of (leaders, cds, trailers)
type	which value to use, "fpkm", alternative "counts".
height	numeric, default 400 (in mm)
width	numeric, default 400 (in mm)
size	numeric, size of dots, default 0.15.

**Value**

invisible(NULL)

---

countOverlapsW	<i>CountOverlaps with weights</i>
----------------	-----------------------------------

---

**Description**

Similar to countOverlaps, but takes an optional weight column. This is usually the score column

**Usage**

```
countOverlapsW(query, subject, weight = NULL, ...)
```

**Arguments**

query	IRanges, IRangesList, GRanges, GRangesList object. Usually transcript a transcript region.
subject	GRanges, GRangesList, GAlignment, usually reads.
weight	(default: NULL), if defined either numeric or character name of valid meta column in subject. If weight is single numeric, it is used for all. A normal weight is the score column given as weight = "score". GRanges("chr1", 1, "+", score = 5), would mean score column tells that this alignment region was found 5 times.
...	additional arguments passed to countOverlaps/findOverlaps

**Value**

a named vector of number of overlaps to subject weighed by 'weight' column.

**See Also**

Other features: [computeFeaturesCage\(\)](#), [computeFeatures\(\)](#), [disengagementScore\(\)](#), [distToCds\(\)](#), [distToTSS\(\)](#), [entropy\(\)](#), [floss\(\)](#), [fpkm\\_calc\(\)](#), [fpkm\(\)](#), [fractionLength\(\)](#), [initiationScore\(\)](#), [insideOutsideORF\(\)](#), [isInFrame\(\)](#), [isOverlapping\(\)](#), [kozakSequenceScore\(\)](#), [orfScore\(\)](#), [rankOrder\(\)](#), [ribosomeReleaseScore\(\)](#), [ribosomeStallingScore\(\)](#), [startRegionCoverage\(\)](#), [startRegion\(\)](#), [stopRegion\(\)](#), [subsetCoverage\(\)](#), [translationalEff\(\)](#)

**Examples**

```
gr1 <- GRanges(seqnames="chr1",
               ranges=IRanges(start = c(4, 9, 10, 30),
                              end = c(4, 15, 20, 31)),
               strand="+")
gr2 <- GRanges(seqnames="chr1",
               ranges=IRanges(start = c(1, 4, 15, 25),
                              end = c(2, 4, 20, 26)),
               strand=c("+"),
               score=c(10, 20, 15, 5))
countOverlaps(gr1, gr2)
countOverlapsW(gr1, gr2, weight = "score")
```

---

countTable

*Extract count table directly from experiment*

---

**Description**

Used to quickly load read count tables to R.

If df is experiment: Extracts by getting /QC\_STATS directory, and searching for region Requires [ORFikQC](#) to have been run on experiment!

**Usage**

```
countTable(df, region = "mrna", type = "count", collapse = FALSE)
```

**Arguments**

df	an <a href="#">ORFik experiment</a> or path to folder with countTable, use path if not same folder as experiment libraries. Will subset to the count tables specified if df is experiment. If experiment has 4 rows and you subset it to only 2, then only those 2 count tables will be outputted.
region	a character vector (default: "mrna"), make raw count matrices of whole mRNAs or one of (leaders, cds, trailers).
type	character, default: "count" (raw counts matrix). Which object type and normalization do you want? "summarized" (SummarizedExperiment object), "deseq" (Deseq2 experiment, design will be all valid non-unique columns except replicates, change by using DESeq2::design, normalization alternatives are: "fpkm", "log2fpkm" or "log10fpkm").

**collapse** a logical/character (default FALSE), if TRUE all samples within the group SAMPLE will be collapsed to one. If "all", all groups will be merged into 1 column called merged\_all. Collapse is defined as  $\text{rowSum}(\text{elements\_per\_group}) / \text{ncol}(\text{elements\_per\_group})$

### Details

If df is path to folder: Loads the the file in that directory with the regex region.rds, where region is what is defined by argument.

### Value

a data.table/SummarizedExperiment/DESeq object of columns as counts / normalized counts per library, column name is name of library. Rownames must be unique for now. Might change.

### See Also

Other countTable: [countTable\\_regions\(\)](#)

### Examples

```
# Make experiment
ORFik.template.experiment()
# Make QC report to get counts ++
# ORFikQC(df)

# Get count Table of mrnas
# countTable(df, "mrna")
# Get count Table of cds
# countTable(df, "cds")
# Get count Table of mrnas as fpkm values
# countTable(df, "mrna", type = "count")
# Get count Table of mrnas with collapsed replicates
# countTable(df, "mrna", collapse = TRUE)
# Get count Table of mrnas as summarizedExperiment
# countTable(df, "mrna", type = "summarized")
# Get count Table of mrnas as DESeq2 object,
# for differential expression analysis
# countTable(df, "mrna", type = "deseq")
```

---

countTable\_regions      *Make a list of count matrices from experiment*

---

### Description

Make a list of count matrices from experiment

### Usage

```
countTable_regions(
  df,
  out.dir = dirname(df$filepath[1]),
  longestPerGene = TRUE,
```

```

geneOrTxNames = "tx",
regions = c("mrna", "leaders", "cds", "trailers"),
type = "count",
weight = "score",
BPPARAM = bpparam()
)

```

### Arguments

df	an ORFik <a href="#">experiment</a>
out.dir	optional output directory, default: <code>dirname(df\$filepath[1])</code> . Will make a folder called "QC_STATS" with all results in this directory.
longestPerGene	a logical (default TRUE), if FALSE all transcript isoforms per gene.
geneOrTxNames	a character vector (default "tx"), should row names keep transcript names ("tx") or change to gene names ("gene")
regions	a character vector, default: <code>c("mrna", "leaders", "cds", "trailers")</code> , make raw count matrices of whole regions specified.
type	default: "count" (raw counts matrix), alternative is "fpkm", "log2fpkm" or "log10fpkm"
weight	numeric or character, a column to score overlaps by. Default "score", will check for a metacolumn called "score" in libraries. If not found, will not use weights.
BPPARAM	how many cores/threads to use? default: <code>bpparam()</code>

### Value

a list of data.table, 1 data.table per region. The regions will be the names the list elements.

### See Also

Other countTable: [countTable\(\)](#)

---

coverageByTranscriptW *coverageByTranscript with weights*

---

### Description

Extends the function with weights, see [coverageByTranscript](#) for original function.

### Usage

```
coverageByTranscriptW(x, transcripts, ignore.strand = FALSE, weight = 1L)
```

### Arguments

x	reads ( <a href="#">GRanges</a> , <a href="#">GAlignments</a> )
transcripts	<a href="#">GRangesList</a>
ignore.strand	a logical (default: FALSE)
weight	a vector (default: 1L), if single number applies for all, else it must be the string name of a defined meta column in "x", that gives number of times a read was found. <code>GRanges("chr1", 1, "+", score = 5)</code> , would mean score column tells that this alignment was found 5 times.

**Value**

Integer Rle of coverage, 1 per transcript

---

coverageGroupings	<i>Get grouping for a coverage table in ORFik</i>
-------------------	---

---

**Description**

Either of two groupings: GF: Gene, fraction FGF: Fraction, position, feature It finds which of these exists, and auto groups

**Usage**

```
coverageGroupings(logicals, grouping = "GF")
```

**Arguments**

logicals	size 2 logical vector, the is.null checks for each column,
grouping	which grouping to perform, default "GF" Gene & Fraction grouping. Alternative "FGF", Fraction & position & feature.

**Details**

Normally not used directly!

**Value**

a quote of the grouping to pass to data.table

---

coverageHeatMap	<i>Create a heatmap of coverage</i>
-----------------	-------------------------------------

---

**Description**

Creates a ggplot representing a heatmap of coverage:

- Rows : Position in region
- Columns : Read length
- Index intensity : (color) coverage scoring per index.

Coverage rows in heat map is fraction, usually fractions is divided into unique read lengths (standard Illumina is 76 unique widths, with some minimum cutoff like 15.) Coverage column in heat map is score, default zscore of counts. These are the relative positions you are plotting to. Like +/- relative to TIS or TSS.

**Usage**

```
coverageHeatMap(
  coverage,
  output = NULL,
  scoring = "zscore",
  legendPos = "right",
  addFracPlot = FALSE,
  xlab = "Position relative to start site",
  ylab = "Protected fragment length",
  colors = "default",
  title = NULL,
  increments.y = "auto",
  gradient.max = max(coverage$score)
)
```

**Arguments**

coverage	a data.table, e.g. output of scaledWindowCoverage
output	character string (NULL), if set, saves the plot as pdf or png to path given. If no format is given, is save as pdf.
scoring	character vector, default "zscore", Which scoring did you use to create? either of zscore, transcriptNormalized, sum, mean, median, .. see ?coverageScorings for info and more alternatives.
legendPos	a character, Default "right". Where should the fill legend be ? ("top", "bottom", "right", "left")
addFracPlot	Add margin histogram plot on top of heatmap with fractions per positions
xlab	the x-axis label, default "Position relative to start site"
ylab	the y-axis label, default "Protected fragment length"
colors	character vector, default: "default", this gives you: c("white", "yellow2", "yellow3", "lightblue", "blue", "navy"), do "high" for more high contrasts, or specify your own colors.
title	a character, default NULL (no title), what is the top title of plot?
increments.y	increments of y axis, default "auto". Or a numeric value < max position & > min position.
gradient.max	numeric, default: max(coverage\$score). What data value should the top color be ? Good to use if you want to compare 2 samples, with the same color intensity, in that case set this value to the max score of the 2 coverage tables.

**Details**

Colors: Remember if you want to change anything like colors, just return the ggplot object, and reassign like: obj + scale\_color\_brewer() etc. Standard colors are:

- 0 reads in whole readlength :gray
- few reads in position :white
- medium reads in position :yellow
- many reads in position :dark blue

**Value**

a ggplot object of the coverage plot, NULL if output is set, then the plot will only be saved to location.

**See Also**

Other heatmaps: [heatMapL\(\)](#), [heatMapRegion\(\)](#), [heatMap\\_single\(\)](#)

Other coveragePlot: [pSitePlot\(\)](#), [savePlot\(\)](#), [windowCoveragePlot\(\)](#)

**Examples**

```
# An ORF
gr1 <- GRangesList(tx1 = GRanges("1", IRanges(1, 6), "+"))
# Ribo-seq reads
range <- IRanges(c(rep(1, 3), 2, 3, rep(4, 2), 5, 6), width = 1 )
reads <- GRanges("1", range, "+")
reads$size <- c(rep(28, 5), rep(29, 4)) # read size
coverage <- windowPerReadLength(gr1, reads = reads, upstream = 0,
                                downstream = 5)

coverageHeatMap(coverage)

# With top sum bar
coverageHeatMap(coverage, addFracPlot = TRUE)
# See vignette for more examples
```

---

coveragePerTiling      *Get coverage per group*

---

**Description**

It tiles each GRangesList group to width 1, and finds hits per position. A range from 1:5 will split into c(1,2,3,4,5) and count hits on each.

**Usage**

```
coveragePerTiling(
  gr1,
  reads,
  is.sorted = FALSE,
  keep.names = TRUE,
  as.data.table = FALSE,
  withFrames = FALSE,
  weight = "score"
)
```



**Arguments**

<code>grl</code>	a <a href="#">GRangesList</a> of 5' utrs, CDS, transcripts, etc.
<code>reads</code>	a <a href="#">GAlignments</a> or <a href="#">GRanges</a> object of RiboSeq, RnaSeq etc. Weights for scoring is default the 'score' column in 'reads'
<code>is.sorted</code>	logical (FALSE), is <code>grl</code> sorted. That is + strand groups in increasing ranges (1,2,3), and - strand groups in decreasing ranges (3,2,1)
<code>keep.names</code>	logical (TRUE), keep names or not.
<code>as.data.table</code>	a logical (FALSE), return as <code>data.table</code> with 2 columns, position and count.
<code>withFrames</code>	a logical (FALSE), only available if <code>as.data.table</code> is TRUE, return the ORF frame, 1,2,3, where position 1 is 1, 2 is 2 and 4 is 1 etc.
<code>weight</code>	(default: 'score'), if defined a character name of valid meta column in subject. <code>GRanges("chr1", 1, "+", score = 5)</code> , would mean score column tells that this alignment region was found 5 times. ORFik .bedo files, contains a score column like this. As do CAGEr CAGE files and many other package formats. You can also assign a score column manually.

**Details**

This is a safer speedup of `coverageByTranscript` from `GenomicFeatures`. It also gives the possibility to return as `data.table`, for faster computations. NOTE: If `reads` contains a `$score` column, it will presume that this is the number of replicates per reads, weights for the `coverage()` function. So delete the score column or set `weight` to something else if this is not wanted.

**Value**

a `RleList`, one integer-`Rle` per group with # of hits per position. Or `data.table` if `as.data.table` is TRUE.

**See Also**

Other `ExtendGenomicRanges`: [asTX\(\)](#), [extendLeaders\(\)](#), [extendTrailers\(\)](#), [reduceKeepAttr\(\)](#), [tile1\(\)](#), [txSeqsFromFa\(\)](#), [windowPerGroup\(\)](#)

**Examples**

```
ORF <- GRanges(seqnames = "1",
              ranges = IRanges(start = c(1, 10, 20),
                              end = c(5, 15, 25)),
              strand = "+")
grl <- GRangesList(tx1_1 = ORF)
RFP <- GRanges("1", IRanges(25, 25), "+")
coveragePerTiling(grl, RFP, is.sorted = TRUE)
# now as data.table with frames
coveragePerTiling(grl, RFP, is.sorted = TRUE, as.data.table = TRUE,
                  withFrames = TRUE)
```

---

coverageScorings      *Add a coverage scoring scheme*

---

### Description

Different scorings and groupings of a coverage representation.

### Usage

```
coverageScorings(coverage, scoring = "zscore")
```

### Arguments

coverage	a data.table containing at least columns (count, position), it is possible to have additional: (genes, fraction, feature)
scoring	a character, one of (zscore, transcriptNormalized, mean, median, sum, log2sum, log10sum, sumLength, meanPos and frameSum, periodic, NULL). More info in details

### Details

Usually output of metaWindow or scaledWindowPositions is input in this function.

Content of coverage data.table: It must contain the count and position columns.

genes column: If you have multiple windows, the genes column must define which gene/transcript grouping the different counts belong to. If there is only a meta window or only 1 gene/transcript, then this column is not needed.

fraction column: If you have coverage of i.e RNA-seq and Ribo-seq, or TCP -seq of large and small subunit, divide into fractions. Like factor(RNA, RFP)

feature column: If gene group is subdivided into parts, like gene is transcripts, and feature column can be c(leader, cds, trailer) etc.

Given a data.table coverage of counts, add a scoring scheme. per: the grouping given, if genes is defined, group by per gene in default scoring.

Scorings:

- zscore (count-windowMean)/windowSD per)
- transcriptNormalized (sum(count / sum of counts per))
- mean (mean(count per))
- median (median(count per))
- sum (count per)
- log2sum (count per)
- log10sum (count per)
- sumLength (count per) / number of windows
- meanPos (mean per position per gene) used in scaledWindowPositions
- sumPos (sum per position per gene) used in scaledWindowPositions
- frameSum (sum per frame per gene) used in ORFScore

- frameSumPerL (sum per frame per read length)
- frameSumPerLG (sum per frame per read length per gene)
- fracPos (fraction of counts per position per gene)
- periodic (Fourier transform periodicity of meta coverage per fraction)
- NULL (no grouping, return input directly)

### Value

a data.table with new scores (size dependent on score used)

### See Also

Other coverage: [metaWindow\(\)](#), [regionPerReadLength\(\)](#), [scaledWindowPositions\(\)](#), [windowPerReadLength\(\)](#)

### Examples

```
dt <- data.table::data.table(count = c(4, 1, 1, 4, 2, 3),
                             position = c(1, 2, 3, 4, 5, 6))
coverageScorings(dt, scoring = "zscore")

# with grouping gene
dt$genes <- c(rep("tx1", 3), rep("tx2", 3))
coverageScorings(dt, scoring = "zscore")
```

---

create.experiment      *Create a ORFik [experiment](#)*

---

### Description

Create information on runs / samples from an experiment as a single R object. By using files in a folder / folders. It will try to make an experiment table with information per sample. There will be several columns you can fill in, most of there it will try to auto-detect. Like if it is RNA-seq or Ribo-seq, Wild type or mutant etc. You will have to fill in the details that were not auto detected. Easiest way to fill in the blanks are in a csv editor like libre Office or excel. Remember that each row (sample) must have a unique combination of values. An extra column called "reverse" is made if there are paired data, like +/- strand wig files.

### Usage

```
create.experiment(
  dir,
  exper,
  saveDir = "~/Bio_data/ORFik_experiments/",
  txdb = "",
  fa = "",
  organism = "",
  pairedEndBam = FALSE,
  viewTemplate = TRUE,
  types = c("bam", "bed", "wig")
)
```

**Arguments**

dir	Which directory / directories to create experiment from
exper	Short name of experiment, max 5 characters long
saveDir	Directory to save experiment csv file, default: "~/Bio_data/ORFik_experiments/" Set to NULL if you don't want to save it to disc.
txdb	A path to gff/gtf file used for libraries
fa	A path to fasta genome/sequences used for libraries, remember the file must have a fasta index too.
organism	character, default: "" (no organism set), scientific name of organism. Homo sapiens, Danio rerio, Rattus norvegicus etc. If you have a SRA metadata csv file, you can set this argument to study\$ScientificName[1], where study is the SRA metadata for all files that was aligned.
pairedEndBam	logical FALSE, else TRUE, or a logical list of TRUE/FALSE per library you see will be included (run first without and check what order the files will come in) 1 paired end file, then two single will be c(T, F, F). If you have a SRA metadata csv file, you can set this argument to study\$LibraryLayout == "PAIRED", where study is the SRA metadata for all files that was aligned.
viewTemplate	run View() on template when finished, default (TRUE)
types	Default (bam, bed, wig), which types of libraries to allow

**Value**

a data.frame, NOTE: this is not a ORFik experiment, only a template for it!

**See Also**

Other ORFik\_experiment: [ORFik.template.experiment\(\)](#), [bamVarName\(\)](#), [experiment-class](#), [filepath\(\)](#), [libraryTypes\(\)](#), [organism.df\(\)](#), [outputLibs\(\)](#), [read.experiment\(\)](#), [save.experiment\(\)](#), [validateExperiments\(\)](#)

**Examples**

```
# 1. Pick directory
dir <- system.file("extdata", "", package = "ORFik")
# 2. Pick an experiment name
exper <- "ORFik"
# 3. Pick .gff/.gtf location
txdb <- system.file("extdata", "annotations.gtf", package = "ORFik")
# 4. Pick fasta genome of organism
fa <- system.file("extdata", "genome.fasta", package = "ORFik")
# 5. Set organism (optional)
org <- "Homo sapiens"

# Create temple not saved on disc yet:
template <- create.experiment(dir = dir, exper, txdb = txdb,
                             saveDir = NULL,
                             fa = fa, organism = org,
                             viewTemplate = FALSE)
## Now fix non-unique rows: either is libre office, microsoft excel, or in R
template$X5[6] <- "heart"
# read experiment (if you set correctly)
df <- read.experiment(template)
```

```
# Save with: save.experiment(df, file = "path/to/save/experiment.csv")

## Create and save experiment directly:
## Default location: "~/Bio_data/ORFik_experiments/"
#template <- create.experiment(dir = dir, exper, txdb = txdb,
#                               fa = fa, organism = org,
#                               viewTemplate = FALSE)
## Custom location
#template <- create.experiment(dir = dir, exper, txdb = txdb,
#                               saveDir = "~/MY/CUSTOME/LOCATION",
#                               fa = fa, organism = org,
#                               viewTemplate = FALSE)
```

---

defineIsoform                      *Overlaps GRanges object with provided annotations.*

---

## Description

Overlaps GRanges object with provided annotations.

## Usage

```
defineIsoform(
  rel_orf,
  tran,
  isoform_names = c("perfect_match", "elong_START_match", "trunc_START_match",
                    "elong_STOP_match", "trunc_STOP_match", "overlap_inside", "overlap_both",
                    "overlap_upstream", "overlap_downstream", "upstream", "downstream", "none")
)
```

## Arguments

rel\_orf                      - GRanges object of your ORF.

tran                         - GRanges object of annotation (transcript or cds) that overlapped in some way rel\_orf.

isoform\_names               - A vector of strings that will be used instead of these defaults: 'perfect\_match' - start and stop matches the tran object strand wise 'elong\_START\_match' - rel\_orf is extension from the STOP side of the tran 'trunc\_START\_match' - rel\_orf is truncation from the STOP side of the tran 'elong\_STOP\_match' - rel\_orf is extension from the START side of the tran 'trunc\_STOP\_match' - rel\_orf is truncation from the START side of the tran 'overlap\_inside' - rel\_orf is inside tran object 'overlap\_both' - rel\_orf contains tran object inside 'overlap\_upstream' - rel\_orf is overlapping upstream part of the tran 'overlap\_downstream' - rel\_orf is overlapping downstream part of the tran 'upstream' - rel\_orf is upstream towards the tran 'downstream' - rel\_orf is downstream towards the tran 'none' - when none of the above options is true

## Value

A string object of defined isoform towards transcript.

---

defineTrailer                      *Defines trailers for ORF.*

---

### Description

Creates GRanges object as a trailer for ORFranges representing ORF, maintaining restrictions of transcriptRanges. Assumes that ORFranges is on the transcriptRanges, strands and seqlevels are in agreement. When lengthOFtrailer is smaller than space left on the transcript than all available space is returned as trailer.

### Usage

```
defineTrailer(ORFranges, transcriptRanges, lengthOftrailer = 200)
```

### Arguments

ORFranges                      GRanges object of your Open Reading Frame.

transcriptRanges  
                                    GRanges object of transtript.

lengthOftrailer  
                                    Numeric. Default is 10.

### Details

It assumes that ORFranges and transcriptRanges are not sorted when on minus strand. Should be like: (200, 600) (50, 100)

### Value

A GRanges object of trailer.

### See Also

Other ORFHelpers: [longestORFs\(\)](#), [mapToGRanges\(\)](#), [orfID\(\)](#), [startCodons\(\)](#), [startSites\(\)](#), [stopCodons\(\)](#), [stopSites\(\)](#), [txNames\(\)](#), [uniqueGroups\(\)](#), [uniqueOrder\(\)](#)

### Examples

```
ORFranges <- GRanges(seqnames = Rle(rep("1", 3)),
                    ranges = IRanges(start = c(1, 10, 20),
                                     end = c(5, 15, 25)),
                    strand = "+")
transcriptRanges <- GRanges(seqnames = Rle(rep("1", 5)),
                           ranges = IRanges(start = c(1, 10, 20, 30, 40),
                                             end = c(5, 15, 25, 35, 45)),
                           strand = "+")
defineTrailer(ORFranges, transcriptRanges)
```

---

detectRibosomeShifts *Detect ribosome shifts*

---

## Description

Utilizes periodicity measurement (Fourier transform), and change point analysis to detect ribosomal footprint shifts for each of the ribosomal read lengths. Returns subset of read lengths and their shifts for which top covered transcripts follow periodicity measure. Each shift value assumes 5' anchoring of the reads, so that output offsets values will shift 5' anchored footprints to be on the p-site of the ribosome. The E-site will be shift + 3 and A site will be shift - 3. So update to these, if you rather want those.

## Usage

```
detectRibosomeShifts(
  footprints,
  txdb,
  start = TRUE,
  stop = FALSE,
  top_tx = 10L,
  minFiveUTR = 30L,
  minCDS = 150L,
  minThreeUTR = 30L,
  firstN = 150L,
  tx = NULL,
  min_reads = 1000,
  accepted.lengths = 26:34,
  heatmap = FALSE,
  must.be.periodic = TRUE
)
```

## Arguments

footprints	<a href="#">GAlignments</a> object of RiboSeq reads - footprints, can also be path to the .bam / .ofst file. If GAlignment object has a meta column called "score", this will be used as replicate numbering for that read. So be careful if you have custom files with score columns, with another meaning.
txdb	a TxDb file, a path to one of: (.gtf, .gff, .gff2, .gff2, .db or .sqlite) or an ORFik experiment
start	(logical) Whether to include predictions based on the start codons. Default TRUE.
stop	(logical) Whether to include predictions based on the stop codons. Default FALSE. Only use if there exists 3' UTRs for the annotation. If periodicity around stop codon is stronger than at the start codon, use stop instead of start region for p-shifting.
top_tx	(integer), default 10. Specify which reads transcripts to use for estimation of the shifts. By default we take top 10 top covered transcripts as they represent less noisy dataset. This is only applicable when there are more than 1000 transcripts.
minFiveUTR	(integer) minimum bp for 5' UTR during filtering for the transcripts. Set to NULL if no 5' UTRs exists for annotation.

minCDS	(integer) minimum bp for CDS during filtering for the transcripts
minThreeUTR	(integer) minimum bp for 3' UTR during filtering for the transcripts. Set to NULL if no 3' UTRs exists for annotation.
firstN	(integer) Represents how many bases of the transcripts downstream of start codons to use for initial estimation of the periodicity.
tx	a GRangesList, if you do not have 5' UTRs in annotation, send your own version. Example: extendLeaders(tx, 30) Where 30 bases will be new "leaders". Since each original transcript was either only CDS or non-coding (filtered out).
min_reads	default (1000), how many reads must a read-length have to be considered for periodicity.
accepted.lengths	accepted readlengths, default 26:34, usually ribo-seq is strongest between 27:32.
heatmap	a logical or character string, default FALSE. If TRUE, will plot heatmap of raw reads before p-shifting to console, to see if shifts given make sense. You can also set a filepath to save the file there.
must.be.periodic	logical TRUE, if FALSE will not filter on periodic read lengths. (The Fourier transform filter will be skipped).

## Details

Check out vignette for the examples of plotting RiboSeq metaplots over start and stop codons, so that you can verify visually whether this function detects correct shifts.

For how the Fourier transform works, see: [isPeriodic](#)

For how the changepoint analysis works, see: [changePointAnalysis](#)

NOTE: It will remove softclips from valid width, the CIGAR 3S30M is qwidth 33, but will remove 3S so final read width is 30 in ORFik. This is standard for ribo-seq.

## Value

a data.table with lengths of footprints and their predicted corresponding offsets

## References

<https://bmcbgenomics.biomedcentral.com/articles/10.1186/s12864-018-4912-6>

## See Also

Other pshifting: [changePointAnalysis\(\)](#), [shiftFootprintsByExperiment\(\)](#), [shiftFootprints\(\)](#)

## Examples

```
# Basic run
#detectRibosomeShifts(footprints, txdb)
# Full example
## Not run:
# Transcriptome annotation ->
gtf_file <- system.file("extdata", "annotations.gtf", package = "ORFik")
# The ribo seq file, usually .bam file ->
riboSeq_file <- system.file("extdata", "ribo-seq.bam", package = "ORFik")
footprints <- GenomicAlignments::readGAlignments(
```



```

    riboSeq_file, param = ScanBamParam(flag = scanBamFlag(
      isDuplicate = FALSE, isSecondaryAlignment = FALSE)))

detectRibosomeShifts(footprints, gtf_file, stop = TRUE)

# Without 5' Annotation
library(GenomicFeatures)

txdb <- loadTxdb(gtf_file)
tx <- exonsBy(txdb, by = "tx", use.names = TRUE)
tx <- extendLeaders(tx, 30)
# Now run function, without 5' and 3' UTRs
detectRibosomeShifts(footprints, txdb, start = TRUE, minFiveUTR = NULL,
                     minCDS = 150L, minThreeUTR = NULL, firstN = 150L,
                     tx = tx)

## End(Not run)

```

---

disengagementScore      *Disengagement score (DS)*

---

## Description

Disengagement score is defined as

$$\frac{\text{(RPFs over ORF)}}{\text{(RPFs downstream to transcript end)}}$$

A pseudo-count of one is added to both the ORF and downstream sums.

## Usage

```

disengagementScore(
  grl,
  RFP,
  GtfOrTx,
  RFP.sorted = FALSE,
  weight = 1L,
  overlapGr1 = NULL
)

```

## Arguments

grl	a <a href="#">GRangesList</a> object with usually either leaders, cds', 3' utrs or ORFs.
RFP	RiboSeq reads as GAlignments, GRanges or GRangesList object
GtfOrTx	If it is <a href="#">TxDb</a> object transcripts will be extracted using exonsBy(Gtf, by = "tx", use.names = TRUE). Else it must be <a href="#">GRangesList</a>
RFP.sorted	logical (FALSE), an optimizer, have you ran this line: RFP <- sort(RFP[countOverlaps(RFP, tx, type = "within") > 0]) Normally not touched, for internal optimization purposes.

weight	a vector (default: 1L, if 1L it is identical to countOverlaps(), if single number (!= 1), it applies for all, if more than one must be equal size of 'reads'. else it must be the string name of a defined meta column in subject "reads", that gives number of times a read was found. GRanges("chr1", 1, "+", score = 5), would mean "score" column tells that this alignment region was found 5 times.
overlapGr1	an integer, (default: NULL), if defined must be countOverlaps(gr1, RFP), added for speed if you already have it

**Value**

a named vector of numeric values of scores

**References**

doi: 10.1242/dev.098344

**See Also**

Other features: [computeFeaturesCage\(\)](#), [computeFeatures\(\)](#), [countOverlapsW\(\)](#), [distToCds\(\)](#), [distToTSS\(\)](#), [entropy\(\)](#), [floss\(\)](#), [fpkm\\_calc\(\)](#), [fpkm\(\)](#), [fractionLength\(\)](#), [initiationScore\(\)](#), [insideOutsideORF\(\)](#), [isInFrame\(\)](#), [isOverlapping\(\)](#), [kozakSequenceScore\(\)](#), [orfScore\(\)](#), [rankOrder\(\)](#), [ribosomeReleaseScore\(\)](#), [ribosomeStallingScore\(\)](#), [startRegionCoverage\(\)](#), [startRegion\(\)](#), [stopRegion\(\)](#), [subsetCoverage\(\)](#), [translationalEff\(\)](#)

**Examples**

```
ORF <- GRanges(seqnames = "1",
               ranges = IRanges(start = c(1, 10, 20), end = c(5, 15, 25)),
               strand = "+")
gr1 <- GRangesList(tx1_1 = ORF)
tx <- GRangesList(tx1 = GRanges("1", IRanges(1, 50), "+"))
RFP <- GRanges("1", IRanges(c(1,10,20,30,40), width = 3), "+")
disengagementScore(gr1, RFP, tx)
```

---

distToCds

*Get distances between ORF ends and starts of their transcripts cds.*

---

**Description**

Will calculate distance between each ORF end and beginning of the corresponding cds (main ORF). Matching is done by transcript names. This is applicable practically to the upstream (fiveUTRs) ORFs only. The cds start site, will be presumed to be on + 1 of end of fiveUTRs.

**Usage**

```
distToCds(ORFs, fiveUTRs, cds = NULL)
```

**Arguments**

ORFs	orfs as <a href="#">GRangesList</a> , names of orfs must be transcript names
fiveUTRs	fiveUTRs as <a href="#">GRangesList</a> , remember to use CAGE version of 5' if you did CAGE reassignment!
cds	cds' as <a href="#">GRangesList</a> , only add if you have ORFs going into CDS.

**Value**

an integer vector, +1 means one base upstream of cds, -1 means 2nd base in cds, 0 means orf stops at cds start.

**References**

doi: 10.1074/jbc.R116.733899

**See Also**

Other features: [computeFeaturesCage\(\)](#), [computeFeatures\(\)](#), [countOverlapsW\(\)](#), [disengagementScore\(\)](#), [distToTSS\(\)](#), [entropy\(\)](#), [floss\(\)](#), [fpkm\\_calc\(\)](#), [fpkm\(\)](#), [fractionLength\(\)](#), [initiationScore\(\)](#), [insideOutsideORF\(\)](#), [isInFrame\(\)](#), [isOverlapping\(\)](#), [kozakSequenceScore\(\)](#), [orfScore\(\)](#), [rankOrder\(\)](#), [ribosomeReleaseScore\(\)](#), [ribosomeStallingScore\(\)](#), [startRegionCoverage\(\)](#), [startRegion\(\)](#), [stopRegion\(\)](#), [subsetCoverage\(\)](#), [translationalEff\(\)](#)

**Examples**

```
gr1 <- GRangesList(tx1_1 = GRanges("1", IRanges(1, 10), "+"))
fiveUTRs <- GRangesList(tx1 = GRanges("1", IRanges(1, 20), "+"))
distToCds(gr1, fiveUTRs)
```

---

distToTSS

*Get distances between ORF Start and TSS of its transcript*


---

**Description**

Matching is done by transcript names. This is applicable practically to any region in Transcript If ORF is not within specified search space in tx, this function will crash.

**Usage**

```
distToTSS(ORFs, tx)
```

**Arguments**

ORFs            orfs as [GRangesList](#), names of orfs must be txname\_[rank]  
tx                transcripts as [GRangesList](#).

**Value**

an integer vector, 1 means on TSS, 2 means second base of Tx.

**References**

doi: 10.1074/jbc.R116.733899

**See Also**

Other features: [computeFeaturesCage\(\)](#), [computeFeatures\(\)](#), [countOverlapsW\(\)](#), [disengagementScore\(\)](#), [distToCds\(\)](#), [entropy\(\)](#), [floss\(\)](#), [fpkm\\_calc\(\)](#), [fpkm\(\)](#), [fractionLength\(\)](#), [initiationScore\(\)](#), [insideOutsideORF\(\)](#), [isInFrame\(\)](#), [isOverlapping\(\)](#), [kozakSequenceScore\(\)](#), [orfScore\(\)](#), [rankOrder\(\)](#), [ribosomeReleaseScore\(\)](#), [ribosomeStallingScore\(\)](#), [startRegionCoverage\(\)](#), [startRegion\(\)](#), [stopRegion\(\)](#), [subsetCoverage\(\)](#), [translationalEff\(\)](#)

**Examples**

```
grl <- GRangesList(tx1_1 = GRanges("1", IRanges(5, 10), "+"))
tx <- GRangesList(tx1 = GRanges("1", IRanges(2, 20), "+"))
distToTSS(grl, tx)
```

---

download.SRA

*Download SRR libraries from SRA*


---

**Description**

Multicore version download, see documentation for SRA toolkit for more information.

**Usage**

```
download.SRA(
  info,
  outdir,
  rename = TRUE,
  fastq.dump.path = install.sratoolkit(),
  settings = paste("--skip-technical", "--split-files"),
  subset = NULL,
  compress = TRUE,
  BPPARAM = bpparam()
)
```

**Arguments**

info	character vector of only SRR numbers or a data.frame with SRA metadata information including the SRR numbers in a column called "Run" or "SRR". Can be SRR, ERR or DRR numbers. If only SRR numbers can not rename, since no additional information is given.
outdir	a string, default: cbu server
rename	logical, default TRUE. Priority of renaming from the metadata is to check for unique names in the LibraryName column, then the sample_title column if no valid names in LibraryName. If new names found and still duplicates, will add "_rep1", "_rep2" to make them unique. If no valid names, will not rename, that is keep the SRR numbers, you then can manually rename files to something more meaningful.
fastq.dump.path	path to fastq-dump binary, default: path returned from install.sratoolkit()

settings	a string of arguments for fastq-dump, default: paste("--gzip", "--skip-technical", "--split-files")
subset	an integer or NULL, default NULL (no subset). If defined as a integer will download only the first n reads specified by subset.
compress	logical, default TRUE. Download compressed files ".gz".
BPPARAM	how many cores/threads to use? default: bpparam(). To see number of threads used, do bpparam()\$workers

**Value**

a character vector of download files filepaths

**References**

<https://ncbi.github.io/sra-tools/fastq-dump.html>

**Examples**

```
SRR <- c("SRR453566") # Can be more than one

## Simple single SRR run of YEAST
outdir <- tempdir() # Specify output directory
# Download, get 5 first reads
download.SRA(SRR, outdir, subset = 5)

## Using metadata column to get SRR numbers and to be able to rename samples
outdir <- tempdir() # Specify output directory
info <- download.SRA.metadata("SRP226389", outdir) # By study id
# Download, 5 first reads of each library and rename
download.SRA(info, outdir, subset = 5)
```

---

download.SRA.metadata *Downloads metadata from SRA*

---

**Description**

Downloads metadata from SRA

**Usage**

```
download.SRA.metadata(SRP, outdir, remove.invalid = TRUE)
```

**Arguments**

SRP	a string, a study ID as either the SRP, ERP, DRP or PRJ of the study, examples would be "SRP226389" or "ERP116106".
outdir	directory to save file, The file will be called "SraRunInfo_SRP.csv", where SRP is the SRP argument. The directory will be created if not existing.
remove.invalid	logical, default TRUE. Remove Runs with 0 reads (spots)

**Value**

a data.table of the opened file

**References**

doi: 10.1093/nar/gkq1019

**Examples**

```
## Originally on SRA
outdir <- tempdir() # Specify output directory
# download.SRA.metadata("SRP226389", outdir)
## Originally on ENA
# download.SRA.metadata("ERP116106", outdir)
```

---

downstreamFromPerGroup

*Get rest of objects downstream (inclusive)*

---

**Description**

Per group get the part downstream of position. `downstreamFromPerGroup(tx, startSites(threeUTRs, asGR = TRUE))` will return the 3' utrs per transcript as `GRangesList`, usually used for interesting parts of the transcripts.

**Usage**

```
downstreamFromPerGroup(tx, downstreamFrom)
```

**Arguments**

`tx` a `GRangesList`, usually of Transcripts to be changed  
`downstreamFrom` a vector of integers, for each group in `tx`, where is the new start point of first valid exon.

**Details**

If you don't want to include the points given in the region, use [downstreamOfPerGroup](#)

**Value**

a `GRangesList` of downstream part

**See Also**

Other `GRanges`: [assignFirstExonsStartSite\(\)](#), [assignLastExonsStopSite\(\)](#), [downstreamOfPerGroup\(\)](#), [upstreamFromPerGroup\(\)](#), [upstreamOfPerGroup\(\)](#)

---

downstreamN	<i>Restrict GRangesList</i>
-------------	-----------------------------

---

**Description**

Will restrict GRangesList to 'N' bp downstream from the first base.

**Usage**

```
downstreamN(grl, firstN = 150L)
```

**Arguments**

grl	(GRangesList)
firstN	(integer) Allow only this many bp downstream, maximum.

**Value**

a GRangesList of reads restricted to firstN and tiled by 1

---

downstreamOfPerGroup	<i>Get rest of objects downstream (exclusive)</i>
----------------------	---

---

**Description**

Per group get the part downstream of position. downstreamOfPerGroup(tx, stopSites(cds, asGR = TRUE)) will return the 3' utrs per transcript as GRangesList, usually used for interesting parts of the transcripts.

**Usage**

```
downstreamOfPerGroup(tx, downstreamOf)
```

**Arguments**

tx	a <a href="#">GRangesList</a> , usually of Transcripts to be changed
downstreamOf	a vector of integers, for each group in tx, where is the new start point of first valid exon. Can also be a GRangesList, then stopsites will be used.

**Details**

If you want to include the points given in the region, use downstreamFromPerGroup

**Value**

a GRangesList of downstream part

**See Also**

Other GRanges: [assignFirstExonsStartSite\(\)](#), [assignLastExonsStopSite\(\)](#), [downstreamFromPerGroup\(\)](#), [upstreamFromPerGroup\(\)](#), [upstreamOfPerGroup\(\)](#)

DTEG.analysis

*Run differential TE analysis***Description**

Using an equal reimplementaion of the deltaTE algorithm (see reference). You need at least 2 groups and 2 replicates per group. The Ribo-seq will be over CDS and RNA-seq over mRNAs, per transcript, subset to genes the returned object, if you do not need isoform variants.

**Usage**

```
DTEG.analysis(
  df.rfp,
  df.rna,
  output.dir = paste0(dirname(df.rfp$filepath[1]), "/QC_STATS/"),
  design = "stage",
  p.value = 0.05,
  plot.title = "",
  width = 6,
  height = 6,
  dot.size = 0.4
)
```

**Arguments**

<code>df.rfp</code>	a <a href="#">experiment</a> of Ribo-seq or 80S from TCP-seq.
<code>df.rna</code>	a <a href="#">experiment</a> of RNA-seq
<code>output.dir</code>	output.dir directory to save plots, plot will be named "TE_between.png". If NULL, will not save.
<code>design</code>	a character vector, default "stage". The columns in the experiment that creates the comparison contrasts.
<code>p.value</code>	a numeric, default 0.05 in interval (0,1) or "" to not show. What p-value used for the analysis? Will be shown as a caption.
<code>plot.title</code>	title for plots, usually name of experiment etc
<code>width</code>	numeric, default 6 (in inches)
<code>height</code>	numeric, default 6 (in inches)
<code>dot.size</code>	numeric, default 0.4, size of point dots in plot.

**Details**

Creates a total of 3 DESeq models (given x is design argument input and libraryType is RNA-seq and Ribo-seq):

1. Ribo-seq model:  $\text{design} = \sim x$  (differences between the x groups in Ribo-seq)
2. RNA-seq model:  $\text{design} = \sim x$  (differences between the x groups in RNA-seq)
3. TE model:  $\text{design} = \sim \text{library type} + x + \text{libraryType} + \text{libraryType}:x$  (differences between the x and libraryType groups and the interaction between them)

What the deltaTE plot calls intensified is here called mRNA abundance and forwarded is called Buffering.



**Value**

a data.table with 8 columns.

**References**

doi: 10.1002/cpmb.108

**See Also**

Other TE: [DTEG.plot\(\)](#), [te.table\(\)](#), [te\\_rna.plot\(\)](#)

**Examples**

```
#df.rfp <- read.experiment("Riboseq")
#df.rna <- read.experiment("RNAseq")
#DTEG.analysis(df.rfp, df.rna)
```

---

DTEG.plot

*Plot DTEG result*


---

**Description**

Plot DTEG result

**Usage**

```
DTEG.plot(
  dt,
  output.dir = NULL,
  p.value = 0.05,
  plot.title = "",
  width = 6,
  height = 6,
  dot.size = 0.4,
  xlim = c(-5, 5),
  ylim = c(-10, 10)
)
```

**Arguments**

dt	a data.table with the results from <a href="#">DTEG.analysis</a>
output.dir	a character path, default NULL(no save), or a directory to save to a file will be called "DTEG_plot.png"
p.value	a numeric, default 0.05 in interval (0,1) or "" to not show. What p-value used for the analysis? Will be shown as a caption.
plot.title	title for plots, usually name of experiment etc
width	numeric, default 6 (in inches)
height	numeric, default 6 (in inches)
dot.size	numeric, default 0.4, size of point dots in plot.
xlim	numeric vector, default c(-5, 5)
ylim	numeric vector, default c(-10, 10)

**Value**

a ggplot object

**See Also**

Other TE: [DTEG.analysis\(\)](#), [te.table\(\)](#), [te\\_rna.plot\(\)](#)

**Examples**

```
#df.rfp <- read.experiment("Riboseq")
#df.rna <- read.experiment("RNAseq")
#dt <- DTEG.analysis(df.rfp, df.rna)
#DTEG.plot(dt)
```

---

entropy

*Percentage of maximum entropy*

---

**Description**

Calculates entropy of the ‘reads’ coverage over each ‘grl’ group. The entropy value per group is a real number in the interval (0:1), where 0 indicates no variance in reads over group. For example  $c(0,0,0,0)$  has 0 entropy, since no reads overlap.

**Usage**

```
entropy(grl, reads, weight = 1L, is.sorted = FALSE, overlapGr1 = NULL)
```

**Arguments**

grl	a <a href="#">GRangesList</a> object can be either transcripts, 5’ utrs, cds’, 3’ utrs or ORFs as a special case (uORFs, potential new cds’ etc). If regions are not spliced you can send a <a href="#">GRanges</a> object.
reads	a <a href="#">GAlignments</a> , <a href="#">GRanges</a> or <a href="#">GRangesList</a> object, usually of RiboSeq, RnaSeq, CageSeq, etc.
weight	a vector (default: 1L, if 1L it is identical to <code>countOverlaps()</code> ), if single number ( $\neq 1$ ), it applies for all, if more than one must be equal size of ‘reads’. else it must be the string name of a defined meta column in subject "reads", that gives number of times a read was found. <code>GRanges("chr1", 1, "+", score = 5)</code> , would mean "score" column tells that this alignment region was found 5 times.
is.sorted	logical (FALSE), is grl sorted. That is + strand groups in increasing ranges (1,2,3), and - strand groups in decreasing ranges (3,2,1)
overlapGr1	an integer, (default: NULL), if defined must be <code>countOverlaps(grl, RFP)</code> , added for speed if you already have it

**Value**

A numeric vector containing one entropy value per element in ‘grl’

**See Also**

Other features: [computeFeaturesCage\(\)](#), [computeFeatures\(\)](#), [countOverlapsW\(\)](#), [disengagementScore\(\)](#), [distToCds\(\)](#), [distToTSS\(\)](#), [floss\(\)](#), [fpkm\\_calc\(\)](#), [fpkm\(\)](#), [fractionLength\(\)](#), [initiationScore\(\)](#), [insideOutsideORF\(\)](#), [isInFrame\(\)](#), [isOverlapping\(\)](#), [kozakSequenceScore\(\)](#), [orfScore\(\)](#), [rankOrder\(\)](#), [ribosomeReleaseScore\(\)](#), [ribosomeStallingScore\(\)](#), [startRegionCoverage\(\)](#), [startRegion\(\)](#), [stopRegion\(\)](#), [subsetCoverage\(\)](#), [translationalEff\(\)](#)

**Examples**

```
# a toy example with ribo-seq p-shifted reads
ORF <- GRanges("1", ranges = IRanges(start = c(1, 12, 22),
                                     end = c(10, 20, 32)),
              strand = "+",
              names = rep("tx1_1", 3))
names(ORF) <- rep("tx1", 3)
gr1 <- GRangesList(tx1_1 = ORF)
reads <- GRanges("1", IRanges(c(25, 35), c(25, 35)), "+")
# gr1 must have same names as cds + _1 etc, so that they can be matched.
entropy(gr1, reads)
# or on cds
cdsORF <- GRanges("1", IRanges(35, 44), "+", names = "tx1")
names(cdsORF) <- "tx1"
cds <- GRangesList(tx1 = cdsORF)
entropy(cds, reads)
```

---

 experiment-class

*experiment class definition*


---

**Description**

It is an object to massively simplify your coding, by having a table of all libraries of an experiment. That contains filepaths and info for each library in the experiment. It also tries to guess grouping / types / pairs by the file names.

Act as a way of extension of [SummarizedExperiment](#) by allowing more ease to find not only counts, but rather information about libraries, and annotation, so that more tasks are possible. Like coverage per position in some transcript etc.

**## Constructor:**

Simplest way to make is to call:

```
create.experiment(dir)
```

On some folder with NGS libraries (usually bam files) and see what you get. Some of the fields might be needed to fill in manually. Each resulting row must be unique (not including filepath, they are always unique), that means if it has replicates then that must be said explicit. And all filepaths must be unique and have files with size > 0.

Here all the columns in the experiment will be described: name (column info): examples

**libtype** library type: rna-seq, ribo-seq, CAGE etc

**stage** stage or tissue: 64cell, Shield, HEK293

**rep** replicate: 1,2,3 etc

**condition** treatment or condition: : WT (wild-type), control, target, mzdicer, starved  
**fraction** fraction of total: 18, 19 (TCP / RCP fractions), or other ways to split library.

**filepath** Full filepath to file

**reverse** optional: 2nd filepath or info, only used if paired files

## Details

Special rules:

Supported:

Single/paired end bam, bed, wig, ofst + compressions of these

The reverse column of the experiments says "paired-end" if bam file. If a pair of wig files, forward and reverse strand, reverse is filepath to '-' strand wig file. Paired forward / reverse wig files, must have same name except `_forward` / `_reverse` in name

Paired end bam, when creating experiment, set `pairedEndBam = c(T, T, T, F)`. For 3 paired end libraries, then one single end.

Naming: Will try to guess naming for tissues / stages, replicates etc. If it finds more than one hit for one file, it will not guess. Always check that it guessed correctly.

## See Also

Other ORFik\_experiment: `ORFik.template.experiment()`, `bamVarName()`, `create.experiment()`, `filepath()`, `libraryTypes()`, `organism.df()`, `outputLibs()`, `read.experiment()`, `save.experiment()`, `validateExperiments()`

## Examples

```
## To see an internal ORFik example
df <- ORFik.template.experiment()
## See libraries in experiment
df
## See organism of experiment
organism.df(df)
## See file paths in experiment
filepath(df, "default")
## Output objects in R, to .GlobalEnv
#outputLibs(df)

## This is how to make it:
## Not run:
library(ORFik)

# 1. Update path to experiment data directory (bam, bed, wig files etc)
exp_dir = "/data/processed_data/RNA-seq/Lee_zebrafish_2013/aligned/"

# 2. Set a 5 character name for experiment, (Lee et al 2013 -> Lee13, etc)
exper_name = "Lee13"

# 3. Create a template experiment (gtf and fasta genome)
temp <- create.experiment(exp_dir, exper_name, saveDir = NULL,
  txdb = "/data/references/Zv9_zebrafish/Danio_rerio.Zv9.79.gtf",
  fa = "/data/references/Zv9_zebrafish/Danio_rerio.Zv9.fa",
  organism = "Homo sapiens")

# 4. Make sure each row(sample) is unique and correct
```

```

# You will get a view open now, check the data.frame that it is correct:
# library type (RNA-seq, Ribo-seq), stage, rep, condition, fraction.
# Let say it did not figure out it is RNA-seq, then we do:"

temp[5:6, 1] <- "RNA" # [row 5 and 6, col 1] are library types

# You can also do this in your spread sheet program (excel, libre office)
# Now save new version, if you did not use spread sheet.
saveName <- paste0("/data/processed_data/experiment_tables_for_R/",
  exper_name, ".csv")
save.experiment(temp, saveName)

# 5. Load experiment, this will validate that you actually made it correct
df <- read.experiment(saveName)

# Set experiment name not to be assigned in R variable names
df@expInVarName <- FALSE
df

## End(Not run)

```

---

experiment.colors      *Decide color for libraries by grouping*

---

## Description

Pick the grouping wanted for colors, by default only group by libtype. Like RNA-seq(skyblue4) and Ribo-seq(orange).

## Usage

```

experiment.colors(
  df,
  color_list = "default",
  skip.libtype = FALSE,
  skip.stage = TRUE,
  skip.replicate = TRUE,
  skip.fraction = TRUE,
  skip.condition = TRUE
)

```

## Arguments

df	an ORFik <a href="#">experiment</a>
color_list	a character vector of colors, default "default". That is the vector c("skyblue4", "orange", "green", "red", "gray", "yellow", "blue", "red2", "orange3"). Picks number of colors needed to make groupings have unique color
skip.libtype	a logical (FALSE), don't include libtype
skip.stage	a logical (FALSE), don't include stage in variable name.
skip.replicate	a logical (FALSE), don't include replicate in variable name.
skip.fraction	a logical (FALSE), don't include fraction
skip.condition	a logical (FALSE), don't include condition in variable name.

**Value**

a character vector of colors

---

export.bed12	<i>Export as bed12 format</i>
--------------	-------------------------------

---

**Description**

bed format for multiple exons per group, as transcripts. Can be use as alternative as a sparse .gff format for ORFs. Can be direct input for ucsc browser or IGV

**Usage**

```
export.bed12(gr1, file, rgb = 0)
```

**Arguments**

gr1	A GRangesList
file	a character path to valid output file name
rgb	integer vector, default (0), either single integer or vector of same size as gr1 to specify groups. It is adviced to not use more than 8 different groups

**Details**

If gr1 has no names, groups will be named 1,2,3,4..

**Value**

NULL (File is saved as .bed)

**See Also**

Other utils: [bedToGR\(\)](#), [convertToOneBasedRanges\(\)](#), [export.wiggle\(\)](#), [fimport\(\)](#), [findFa\(\)](#), [fread.bed\(\)](#), [optimizeReads\(\)](#), [readBam\(\)](#), [readWig\(\)](#)

**Examples**

```
gr1 <- GRangesList(GRanges("1", c(1,3,5), "+"))
# export.bed12(gr1, "output/path/orfs.bed")
```

---

export.bedo	<i>Store GRanges object as .bedo</i>
-------------	--------------------------------------

---

### Description

.bedo is .bed ORFik, an optimized bed format for coverage reads with read lengths .bedo is a text based format with columns (6 maximum):

1. chromosome
2. start
3. end
4. strand
5. ref width (cigar # M's, match/mismatch total)
6. duplicates of that read

### Usage

```
export.bedo(object, out)
```

### Arguments

object	a GRanges object
out	a character, location on disc (full path)

### Details

Positions are 1-based, not 0-based as .bed. End will be removed if all ends equals all starts. Import with import.bedo

### Value

NULL, object saved to disc

---

export.bedoc	<i>Store GAlignments object as .bedoc</i>
--------------	---

---

### Description

A much faster way to store, load and use bam files.

.bedoc is .bed ORFik, an optimized bed format for coverage reads with cigar and replicate number. .bedoc is a text based format with columns (5 maximum):

1. chromosome
2. cigar: (cigar # M's, match/mismatch total)
3. start (left most position)
4. strand (+, -, \*)
5. score: duplicates of that read

**Usage**

```
export.bedoc(object, out)
```

**Arguments**

```
object      a GAlignments object
out         a character, location on disc (full path)
```

**Details**

Positions are 1-based, not 0-based as .bed. Import with import.bedoc

**Value**

NULL, object saved to disc

---

export.ofst	<i>Store GRanges / GAlignments object as .ofst</i>
-------------	--

---

**Description**

A much faster way to store, load and use bam files.

.ofst is ORFik fast serialized object, an optimized format for coverage reads with cigar and replicate number. It uses the fst format as back-end: [fst-package](#).

A .ofst ribo seq file can compress the information in a bam file from 5GB down to a few MB. This new files has super fast reading time, only a few seconds, instead of minutes. It also has random index access possibility of the file.

.ofst is represented as a data.frame format with minimum 4 columns:

1. chromosome
2. start (left most position)
3. strand (+, -, \*)
4. width (not added if cigar exists)
5. cigar (not needed if width exists): (cigar # M's, match/mismatch total)
5. score: duplicates of that read
6. size: qwidth according to reference of read

If file is from GAlignmentPairs, it will contain a cigar1, cigar2 instead of cigar and start1 and start2 instead of start

**Usage**

```
export.ofst(x, ...)
```

**Arguments**

```
x           a GRanges, GAlignments or GAlignmentPairs object
...         additional arguments for write_fst
```

**Details**

Other columns can be named whatever you want and added to meta columns. Positions are 1-based, not 0-based as .bed. Import with import.ofst



**Value**

NULL, object saved to disc

**Examples**

```
## GRanges
gr <- GRanges("1:1-3:-")
# export.ofst(gr, file = "path.ofst")
## GAlignment
# Make input data.frame
df <- data.frame(seqnames = "1", cigar = "3M", start = 1L, strand = "+")
ga <- ORFik::getGAlignments(df)
# export.ofst(ga, file = "path.ofst")
```

---

export.ofst,GAlignmentPairs-method

*Store GRanges / GAlignments object as .ofst*

---

**Description**

A much faster way to store, load and use bam files.

.ofst is ORFik fast serialized object, an optimized format for coverage reads with cigar and replicate number. It uses the fst format as back-end: [fst-package](#).

A .ofst ribo seq file can compress the information in a bam file from 5GB down to a few MB. This new files has super fast reading time, only a few seconds, instead of minutes. It also has random index access possibility of the file.

.ofst is represented as a data.frame format with minimum 4 columns:

1. chromosome
2. start (left most position)
3. strand (+, -, \*)
4. width (not added if cigar exists)
5. cigar (not needed if width exists): (cigar # M's, match/mismatch total)
5. score: duplicates of that read
6. size: qwidth according to reference of read

If file is from GAlignmentPairs, it will contain a cigar1, cigar2 instead of cigar and start1 and start2 instead of start

**Usage**

```
## S4 method for signature 'GAlignmentPairs'
export.ofst(x, file, ...)
```

**Arguments**

x	a GRanges, GAlignments or GAlignmentPairs object
file	a character, location on disc (full path)
...	additional arguments for write_fst

**Details**

Other columns can be named whatever you want and added to meta columns. Positions are 1-based, not 0-based as .bed. Import with import.ofst

**Value**

NULL, object saved to disc

**Examples**

```
## GRanges
gr <- GRanges("1:1-3:-")
# export.ofst(gr, file = "path.ofst")
## GAlignment
# Make input data.frame
df <- data.frame(seqnames = "1", cigar = "3M", start = 1L, strand = "+")
ga <- ORFik::getGAlignments(df)
# export.ofst(ga, file = "path.ofst")
```

---

export.ofst,GAlignments-method

*Store GRanges / GAlignments object as .ofst*

---

**Description**

A much faster way to store, load and use bam files.

.ofst is ORFik fast serialized object, an optimized format for coverage reads with cigar and replicate number. It uses the fst format as back-end: [fst-package](#).

A .ofst ribo seq file can compress the information in a bam file from 5GB down to a few MB. This new files has super fast reading time, only a few seconds, instead of minutes. It also has random index access possibility of the file.

.ofst is represented as a data.frame format with minimum 4 columns:

1. chromosome
2. start (left most position)
3. strand (+, -, \*)
4. width (not added if cigar exists)
5. cigar (not needed if width exists): (cigar # M's, match/mismatch total)
5. score: duplicates of that read
6. size: qwidth according to reference of read

If file is from GAlignmentPairs, it will contain a cigar1, cigar2 instead of cigar and start1 and start2 instead of start

**Usage**

```
## S4 method for signature 'GAlignments'
export.ofst(x, file, ...)
```

**Arguments**

x	a GRanges, GAlignments or GAlignmentPairs object
file	a character, location on disc (full path)
...	additional arguments for write_fst

**Details**

Other columns can be named whatever you want and added to meta columns. Positions are 1-based, not 0-based as .bed. Import with import.ofst

**Value**

NULL, object saved to disc

**Examples**

```
## GRanges
gr <- GRanges("1:1-3:-")
# export.ofst(gr, file = "path.ofst")
## GAlignment
# Make input data.frame
df <- data.frame(seqnames = "1", cigar = "3M", start = 1L, strand = "+")
ga <- ORFik::getGAlignments(df)
# export.ofst(ga, file = "path.ofst")
```

---

export.ofst,GRanges-method

*Store GRanges / GAlignments object as .ofst*

---

**Description**

A much faster way to store, load and use bam files.

.ofst is ORFik fast serialized object, an optimized format for coverage reads with cigar and replicate number. It uses the fst format as back-end: [fst-package](#).

A .ofst ribo seq file can compress the information in a bam file from 5GB down to a few MB. This new files has super fast reading time, only a few seconds, instead of minutes. It also has random index access possibility of the file.

.ofst is represented as a data.frame format with minimum 4 columns:

1. chromosome
2. start (left most position)
3. strand (+, -, \*)
4. width (not added if cigar exists)
5. cigar (not needed if width exists): (cigar # M's, match/mismatch total)
5. score: duplicates of that read
6. size: qwidth according to reference of read

If file is from GAlignmentPairs, it will contain a cigar1, cigar2 instead of cigar and start1 and start2 instead of start

**Usage**

```
## S4 method for signature 'GRanges'
export.ofst(x, file, ...)
```

**Arguments**

x	a GRanges, GAlignments or GAlignmentPairs object
file	a character, location on disc (full path)
...	additional arguments for write_fst

**Details**

Other columns can be named whatever you want and added to meta columns. Positions are 1-based, not 0-based as .bed. Import with import.ofst

**Value**

NULL, object saved to disc

**Examples**

```
## GRanges
gr <- GRanges("1:1-3:-")
# export.ofst(gr, file = "path.ofst")
## GAlignment
# Make input data.frame
df <- data.frame(seqnames = "1", cigar = "3M", start = 1L, strand = "+")
ga <- ORFik::getGAlignments(df)
# export.ofst(ga, file = "path.ofst")
```

---

export.wiggle

*Export as wiggle format*

---

**Description**

Will create 2 files, 1 for + strand (\*\_forward.wig) and 1 for - strand (\*\_reverse.wig). If all files are \* stranded, will output 1 file. Can be direct input for ucsc browser or IGV

**Usage**

```
export.wiggle(x, file)
```

**Arguments**

x	A GRangesList, GAlignment GAlignmentPairs with score column. Will be converted to 5' end position of original range. If score column does not exist, will group ranges and give replicates as score column.
file	a character path to valid output file name

**Value**

invisible(NULL) (File is saved as 2 .wig files)

**References**

<https://genome.ucsc.edu/goldenPath/help/wiggle.html>

**See Also**

Other utils: [bedToGR\(\)](#), [convertToOneBasedRanges\(\)](#), [export.bed12\(\)](#), [fimport\(\)](#), [findFa\(\)](#), [fread.bed\(\)](#), [optimizeReads\(\)](#), [readBam\(\)](#), [readWig\(\)](#)

**Examples**

```
x <- c(GRanges("1", c(1,3,5), "-"), GRanges("1", c(1,3,5), "+"))
# export.wiggle(x, "output/path/rna.wig")
```

---

 extendLeaders

*Extend the leaders transcription start sites.*


---

**Description**

Will extend the leaders or transcripts upstream (5' end) by extension. The extension is general not relative, that means splicing will not be taken into account. Requires the grl to be sorted beforehand, use [sortPerGroup](#) to get sorted grl.

**Usage**

```
extendLeaders(grl, extension = 1000L, cds = NULL)
```

**Arguments**

grl	usually a <a href="#">GRangesList</a> of 5' utrs or transcripts. Can be used for any extension of groups.
extension	an integer, how much to extend upstream (5' end). Either single value that will apply for all, or same as length of grl which will give 1 update value per grl object. Or a <a href="#">GRangesList</a> where start / stops by strand are the positions to use as new starts.
cds	a <a href="#">GRangesList</a> of coding sequences, If you want to extend 5' leaders downstream, to catch upstream ORFs going into cds, include it. It will add first cds exon to grl matched by names. Do not add for transcripts, as they are already included.

**Value**

an extended [GRangeslist](#)

**See Also**

Other [ExtendGenomicRanges](#): [asTX\(\)](#), [coveragePerTiling\(\)](#), [extendTrailers\(\)](#), [reduceKeepAttr\(\)](#), [tile1\(\)](#), [txSeqsFromFa\(\)](#), [windowPerGroup\(\)](#)

**Examples**

```
library(GenomicFeatures)
samplefile <- system.file("extdata", "hg19_knownGene_sample.sqlite",
                          package = "GenomicFeatures")
txdb <- loadDb(samplefile)
fiveUTRs <- fiveUTRsByTranscript(txdb) # <- extract only 5' leaders
tx <- exonsBy(txdb, by = "tx", use.names = TRUE)
cds <- cdsBy(txdb, "tx", use.names = TRUE)
## now try(extend upstream 1000, downstream 1st cds exons):
extendLeaders(fiveUTRs, extension = 1000, cds)

## when extending transcripts, don't include cds' of course,
```

```
## since they are already there
extendLeaders(tx, extension = 1000)
```

---

extendsTSSexons      *Extend first exon of each transcript with length specified*

---

### Description

Extend first exon of each transcript with length specified

### Usage

```
extendsTSSexons(fiveUTRs, extension = 1000)
```

### Arguments

fiveUTRs      The 5' leader sequences as GRangesList  
 extension      The number of bases to extend transcripts upstream

### Value

GRangesList object of fiveUTRs

---

extendTrailers      *Extend the Trailers transcription stop sites*

---

### Description

Will extend the trailers or transcripts downstream (3' end) by extension. Remember the extension is general not relative, that means splicing will not be taken into account. Requires the grl to be sorted beforehand, use [sortPerGroup](#) to get sorted grl.

### Usage

```
extendTrailers(grl, extension = 1000L)
```

### Arguments

grl      usually a [GRangesList](#) of 3' utrs or transcripts. Can be used for any extension of groups.  
 extension      an integer, how much to extend downstream (3' end). Eiter single value that will apply for all, or same as length of grl which will give 1 update value per grl object. Or a GRangesList where start / stops sites by strand are the positions to use as new starts.

### Value

an extended GRangeslist

**See Also**

Other ExtendGenomicRanges: [asTX\(\)](#), [coveragePerTiling\(\)](#), [extendLeaders\(\)](#), [reduceKeepAttr\(\)](#), [tile1\(\)](#), [txSeqsFromFa\(\)](#), [windowPerGroup\(\)](#)

**Examples**

```
library(GenomicFeatures)
samplefile <- system.file("extdata", "hg19_knownGene_sample.sqlite",
                          package = "GenomicFeatures")
txdb <- loadDb(samplefile)
threeUTRs <- threeUTRsByTranscript(txdb) # <- extract only 5' leaders
tx <- exonsBy(txdb, by = "tx", use.names = TRUE)
## now try(extend downstream 1000):
extendTrailers(threeUTRs, extension = 1000)
## Or on transcripts
extendTrailers(tx, extension = 1000)
```

---

 filepath

*Get filepaths to ORFik experiment*


---

**Description**

If other type than "default" is given and that type is not found, it will return you default filepaths without warning.

**Usage**

```
filepath(df, type, basename = FALSE)
```

**Arguments**

df	an ORFik <a href="#">experiment</a>
type	a character(default: "default"), load files in experiment or some precomputed variant, either "bedo", "bedoc", "ofst or "pshifted". These are made with ORFik:::simpleLibs(), shiftFootprintsByExperiment()..
basename	logical, default (FALSE). Get relative paths instead of full. Only use for inspection!

**Details**

For pshifted libraries, it will load ".bedo" prioritized over ".bed", if there exists both file types for the same file.

**Value**

a character vector of paths, or a list of character with 2 paths per, if paired libraries exists

**See Also**

Other ORFik\_experiment: `ORFik.template.experiment()`, `bamVarName()`, `create.experiment()`, `experiment-class`, `libraryTypes()`, `organism.df()`, `outputLibs()`, `read.experiment()`, `save.experiment()`, `validateExperiments()`

**Examples**

```
df <- ORFik.template.experiment()
filepath(df, "default")
# If you have bedo files, see simpleLibs():
# filepath(df, "bedo")
# If you have pshifted files, see shiftFootprintsByExperiment():
# filepath(df, "pshifted")
```

---

<code>filterCage</code>	<i>Filter peak of cage-data by value</i>
-------------------------	--

---

**Description**

Filter peak of cage-data by value

**Usage**

```
filterCage(cage, filterValue = 1, fiveUTRs = NULL, preCleanup = TRUE)
```

**Arguments**

<code>cage</code>	Either a filePath for the CageSeq file as .bed .bam or .wig, with possible compressions (".gzip", ".gz", ".bgz"), or already loaded CageSeq peak data as GRanges or GAlignment. NOTE: If it is a .bam file, it will add a score column by running: <code>convertToOneBasedRanges(cage, method = "5prime", addScoreColumn = TRUE)</code> The score column is then number of replicates of read, if score column is something else, like read length, set the score column to NULL first.
<code>filterValue</code>	The minimum number of reads on cage position, for it to be counted as possible new tss. (represented in score column in CageSeq data) If you already filtered, set it to 0.
<code>fiveUTRs</code>	a GRangesList (NULL), if added will filter out cage reads by these following rules: all reads in region (-5:-1, 1:5) for each tss will be removed, removes noise.
<code>preCleanup</code>	logical (TRUE), if TRUE, remove all reads in region (-5:-1, 1:5) of all original tss in leaders. This is to keep original TSS if it is only +/- 5 bases from the original.

**Value**

the filtered Granges object



---

 filterExtremePeakGenes

*Filter out transcript by a median filter*


---

### Description

For removing very extreme peaks in coverage plots, use high quantiles, like 99. Used to make your plots look better, by removing extreme peaks.

### Usage

```
filterExtremePeakGenes(
  tx,
  reads,
  upstream = NULL,
  downstream = NULL,
  multiplier = "0.99",
  min_cutoff = "0.999",
  pre_filter_minimum = 0,
  average = "median"
)
```

### Arguments

tx	a GRangesList
reads	a GAlignments or GRanges
upstream	numeric or NULL, default NULL. if you want window of tx, instead of whole, specify how much upstream from start of tx, 10 is include 10 bases before start
downstream	numeric or NULL, default NULL. if you want window of tx, instead of whole, specify how much downstream from start of tx, 10 is go 10 bases into tx from start.
multiplier	a character or numeric, default "0.99", either a quantile if input is string[0-1], like "0.99", or numeric value if input is numeric. How much bigger than median / mean counts per gene, must a value be to be defined as extreme ?
min_cutoff	a character or numeric, default "0.999", either a quantile if input is string[0-1], like "0.999", or numeric value if input is numeric. Lowest allowed value
pre_filter_minimum	numeric, default 0. If value is x, will remove all positions in all genes with coverage < x, before median filter is applied. Set to 1 to remove all 0 positions.
average	character, default "median". Alternative: "mean". How to scale the multiplier argument, from median or mean of gene coverage.

### Value

GRangesList (filtered)

---

filterTranscripts      *Filter transcripts by lengths*

---

### Description

Filter transcripts to those who have leaders, CDS, trailers of some lengths, you can also pick the longest per gene.

### Usage

```
filterTranscripts(
  txdb,
  minFiveUTR = 30L,
  minCDS = 150L,
  minThreeUTR = 30L,
  longestPerGene = TRUE,
  stopOnEmpty = TRUE,
  by = "tx"
)
```

### Arguments

txdb	a TxDb file or a path to one of: (.gtf, .gff, .gff2, .db or .sqlite), if it is a GRangesList, it will return it self.
minFiveUTR	(integer) minimum bp for 5' UTR during filtering for the transcripts. Set to NULL if no 5' UTRs exists for annotation.
minCDS	(integer) minimum bp for CDS during filtering for the transcripts
minThreeUTR	(integer) minimum bp for 3' UTR during filtering for the transcripts. Set to NULL if no 3' UTRs exists for annotation.
longestPerGene	logical (TRUE), return only longest valid transcript per gene. NOTE: This is by priority longest cds isoform, if equal then pick longest total transcript. So if transcript is shorter but cds is longer, it will still be the one returned.
stopOnEmpty	logical TRUE, stop if no valid transcripts are found ?
by	a character, default "tx" Either "tx" or "gene". What names to output region by, the transcript name "tx" or gene names "gene"

### Details

If a transcript does not have a trailer, then the length is 0, so they will be filtered out if you set minThreeUTR to 1. So only transcripts with leaders, cds and trailers will be returned. You can set the integer to 0, that will return all within that group.

If your annotation does not have leaders or trailers, set them to NULL, since 0 does mean there must exist a column called utr3\_len etc. Genes with gene\_id = NA will be removed.

### Value

a character vector of valid transcript names

**Examples**

```
gtf_file <- system.file("extdata", "annotations.gtf", package = "ORFik")
txdb <- GenomicFeatures::makeTxDbFromGFF(gtf_file)
txNames <- filterTranscripts(txdb, minFiveUTR = 1, minCDS = 30,
                             minThreeUTR = 1)
loadRegion(txdb, "mrna")[txNames]
loadRegion(txdb, "5utr")[txNames]
```

---

filterUORFs	<i>Remove uORFs that are false CDS hits</i>
-------------	---

---

**Description**

This is a strong filtering, so that even if the cds is on another transcript , the uORF is filtered out, this is because there is no way of knowing by current ribo-seq, rna-seq experiments.

**Usage**

```
filterUORFs(uorfs, cds)
```

**Arguments**

uorfs (GRangesList), the uORFs to filter  
 cds (GRangesList), the coding sequences (main ORFs on transcripts), to filter against.

**Value**

(GRangesList) of filtered uORFs

**See Also**

Other uorfs: [addCdsOnLeaderEnds\(\)](#), [removeORFsWithSameStartAsCDS\(\)](#), [removeORFsWithSameStopAsCDS\(\)](#), [removeORFsWithStartInsideCDS\(\)](#), [removeORFsWithinCDS\(\)](#), [uORFSearchSpace\(\)](#)

---

fimport	<i>Load any type of sequencing reads</i>
---------	--

---

**Description**

Wraps around `rtracklayer::import` and tries to speed up loading with the use of `data.table`. Supports `gzip`, `gz`, `bgz` compression formats. Also safer chromosome naming with the argument `chrStyle`

**Usage**

```
fimport(path, chrStyle = NULL)
```

**Arguments**

path	a character path to file (1 or 2 files), or data.table with 2 columns(forward&reverse) or a GRanges/Galignment/GAlignmentPairs object etc. If it is ranged object it will presume to be already loaded, so will return the object as it is, updating the seqlevelsStyle if given.
chrStyle	a GRanges object, TxDb, FaFile, or a <a href="#">seqlevelsStyle</a> (Default: NULL) to get seqlevelsStyle from. Is chromosome 1 called chr1 or 1, is mitochondrial chromosome called MT or chrM etc. Will use 1st seqlevel-style if more are present. Like: c("NCBI", "UCSC") -> pick "NCBI"

**Details**

NOTE: For wig you can send in 2 files, so that it automatically merges forward and reverse stranded objects. You can also just send 1 wig file, it will then have "\*" as strand.

**Value**

a [GAlignments/GRanges](#) object, depending on input.

**See Also**

Other utils: [bedToGR\(\)](#), [convertToOneBasedRanges\(\)](#), [export.bed12\(\)](#), [export.wiggle\(\)](#), [findFa\(\)](#), [fread.bed\(\)](#), [optimizeReads\(\)](#), [readBam\(\)](#), [readWig\(\)](#)

**Examples**

```
bam_file <- system.file("extdata", "ribo-seq.bam", package = "ORFik")
fimport(bam_file)
# Certain chromosome naming
fimport(bam_file, "NCBI")
```

---

findFa

*Convenience wrapper for Rsamtools FaFile*

---

**Description**

Get fasta file object, to find sequences in file.  
Will load and import file if necessary.

**Usage**

```
findFa(faFile)
```

**Arguments**

faFile	<a href="#">FaFile</a> , BSgenome, fasta/index file path or an ORFik <a href="#">experiment</a> . This file is usually used to find the transcript sequences from some GRangesList.
--------	---

**Value**

a [FaFile](#) or BSgenome

**See Also**

Other utils: [bedToGR\(\)](#), [convertToOneBasedRanges\(\)](#), [export.bed12\(\)](#), [export.wiggle\(\)](#), [fimport\(\)](#), [fread.bed\(\)](#), [optimizeReads\(\)](#), [readBam\(\)](#), [readWig\(\)](#)

**Examples**

```
# Some fasta genome with existing fasta index in same folder
path <- system.file("extdata", "genome.fasta", package = "ORFik")
findFa(path)
```

---

findFromPath	<i>Find all candidate library types filenames</i>
--------------	---

---

**Description**

From the given [experiment](#)

**Usage**

```
findFromPath(filepaths, candidates)
```

**Arguments**

filepaths	path to all files
candidates	a data.table with 2 columns, Possible names to search for, see <a href="#">experiment_naming</a> family for candidates.

**Value**

a candidate library types (character vector)

---

findLibrariesInFolder	<i>Get all library files in folder/folders of given types</i>
-----------------------	---

---

**Description**

Will try to guess paired / unpaired wig, bed, bam files.

**Usage**

```
findLibrariesInFolder(dir, types, pairedEndBam = FALSE)
```

**Arguments**

dir	Which directory / directories to create experiment from
types	Default (bam, bed, wig), which types of libraries to allow
pairedEndBam	logical FALSE, else TRUE, or a logical list of TRUE/FALSE per library you see will be included (run first without and check what order the files will come in) 1 paired end file, then two single will be c(T, F, F). If you have a SRA metadata csv file, you can set this argument to <code>study\$LibraryLayout == "PAIRED"</code> , where study is the SRA metadata for all files that was aligned.

**Details**

Set pairedEndBam if you have paired end reads as a single bam file.

**Value**

(data.table) All files found from types parameter. With 2 extra column (logical), is it wig pairs, and paired bam files.

---

findMapORFs	<i>Find ORFs and immediately map them to their genomic positions.</i>
-------------	---

---

**Description**

Finds ORFs on the sequences of interest, but returns relative positions to the positions of 'grl' argument. For example, 'grl' can be exons of known transcripts (with genomic coordinates), and 'seq' sequences of those transcripts, in that case, this function will return genomic coordinates of ORFs found on transcript sequences.

**Usage**

```
findMapORFs(
  grl,
  seqs,
  startCodon = startDefinition(1),
  stopCodon = stopDefinition(1),
  longestORF = TRUE,
  minimumLength = 0,
  groupByTx = TRUE
)
```

**Arguments**

grl	(GRangesList) of sequences to search for ORFs, probably in genomic coordinates
seqs	(DNAStrngSet or character vector) - DNA/RNA sequences to search for Open Reading Frames. Can be both uppercase or lowercase. Easiest call to get seqs if you want only regions from a fasta/fastx index pair is: seqs = ORFik:::txSeqsFromFa(grl, faFile), where grl is a GRanges/List of search regions and faFile is a FaFile.
startCodon	(character vector) Possible START codons to search for. Check <a href="#">startDefinition</a> for helper function.
stopCodon	(character vector) Possible STOP codons to search for. Check <a href="#">stopDefinition</a> for helper function.
longestORF	(logical) Default TRUE. Keep only the longest ORF per unique (seqname, strand, stopcodon) combination, you can also use function <a href="#">longestORFs</a> after creation of ORFs for same result.
minimumLength	(integer) Default is 0. Which is START + STOP = 6 bp. Minimum length of ORF, without counting 3bps for START and STOP codons. For example minimumLength = 8 will result in size of ORFs to be at least START + 8*3 (bp) + STOP = 30 bases. Use this param to restrict search.

groupByTx      logical (default: TRUE), should output GRangesList be grouped by orfs per transcript (TRUE) or by exons per ORF (FALSE)?

### Details

This function assumes that 'seq' is in widths relative to 'grl', and that their orders match. 1st seq is 1st grl object, etc.

See vignette for real life example.

### Value

A GRangesList of ORFs.

### See Also

Other findORFs: `findORFsFasta()`, `findORFs()`, `findUORFs()`, `startDefinition()`, `stopDefinition()`

### Examples

```
# This sequence has ORFs at 1-9 and 4-9
seqs <- c("ATGATGTAA") # the dna sequence
findORFs(seqs)
# lets assume that this sequence comes from two exons as follows
gr <- GRanges(seqnames = rep("1", 2), # chromosome 1
              ranges = IRanges(start = c(21, 10), end = c(23, 15)),
              strand = rep("-", 2), names = rep("tx1", 2))
grl <- GRangesList(tx1 = gr)
findMapORFs(grl, seqs) # ORFs are properly mapped to its genomic coordinates

grl <- c(grl, grl)
names(grl) <- c("tx1", "tx2")
findMapORFs(grl, c(seqs, seqs))
```

---

findMaxPeaks	<i>Find max peak for each transcript, returns as data.table, without names, but with index</i>
--------------	--

---

### Description

Find max peak for each transcript, returns as data.table, without names, but with index

### Usage

```
findMaxPeaks(cageOverlaps, filteredCage)
```

### Arguments

cageOverlaps      The cageOverlaps between cage and extended 5' leaders  
 filteredCage      The filtered raw cage-data used to reassign 5' leaders

### Value

a data.table of max peaks

---

findNewTSS	<i>Finds max peaks per transcript from reads in the cagefile</i>
------------	--

---

**Description**

Finds max peaks per transcript from reads in the cagefile

**Usage**

```
findNewTSS(fiveUTRs, cageData, extension, restrictUpstreamToTx)
```

**Arguments**

fiveUTRs	The 5' leader sequences as GRangesList
cageData	The CAGE as GRanges object
extension	The number of bases to extends transcripts upstream.
restrictUpstreamToTx	a logical (FALSE), if you want to restrict leaders to not extend closer than 5 bases from closest upstream leader, set this to TRUE.

**Value**

a Hits object

---

findNGSPairs	<i>Find pair of forward and reverse strand wig / bed files and paired end bam files split in two</i>
--------------	--

---

**Description**

Find pair of forward and reverse strand wig / bed files and paired end bam files split in two

**Usage**

```
findNGSPairs(
  paths,
  f = c("forward", "fwd"),
  r = c("reverse", "rev"),
  format = "wig"
)
```

**Arguments**

paths	a character path at least one .wig / .bed file
f	Default (c("forward", "fwd")) a character vector for forward direction regex.
r	Default (c("reverse", "rev")) a character vector for reverse direction regex.
format	default "wig", for bed do "bed". Also searches compressions of these variants.



**Value**

if not all are paired, return original list, if they are all paired, return a data.table with matches as 2 columns

---

findORFs *Find Open Reading Frames.*

---

**Description**

Find all Open Reading Frames (ORFs) on the input sequences in ONLY 5'-3' direction (+), but within all three possible reading frames. For each sequence of the input vector [IRanges](#) with START and STOP positions (inclusive) will be returned as [IRangesList](#). Returned coordinates are relative to the input sequences.

**Usage**

```
findORFs(
  seqs,
  startCodon = startDefinition(1),
  stopCodon = stopDefinition(1),
  longestORF = TRUE,
  minimumLength = 0
)
```

**Arguments**

seqs	(DNASTringSet or character vector) - DNA/RNA sequences to search for Open Reading Frames. Can be both uppercase or lowercase. Easiest call to get seqs if you want only regions from a fasta/fastq index pair is: seqs = ORFik:::txSeqsFromFa(grl, faFile), where grl is a GRanges/List of search regions and faFile is a <a href="#">FaFile</a> .
startCodon	(character vector) Possible START codons to search for. Check <a href="#">startDefinition</a> for helper function.
stopCodon	(character vector) Possible STOP codons to search for. Check <a href="#">stopDefinition</a> for helper function.
longestORF	(logical) Default TRUE. Keep only the longest ORF per unique (seqname, strand, stopcodon) combination, you can also use function <a href="#">longestORFs</a> after creation of ORFs for same result.
minimumLength	(integer) Default is 0. Which is START + STOP = 6 bp. Minimum length of ORF, without counting 3bps for START and STOP codons. For example minimumLength = 8 will result in size of ORFs to be at least START + 8*3 (bp) + STOP = 30 bases. Use this param to restrict search.

**Details**

If you want antisense strand too, do: `#positive strands pos <- findORFs(seqs) #negative strands (DNASTringSet only if character) neg <- findORFs(reverseComplement(DNASTringSet(seqs))) relist(c(GRanges(pos, strand = "+"), GRanges(neg, strand = "-")), skeleton = merge(pos, neg))`

**Value**

(IRangesList) of ORFs locations by START and STOP sites grouped by input sequences. In a list of sequences, only the indices of the sequences that had ORFs will be returned, e.g. 3 sequences where only 1 and 3 has ORFs, will return size 2 IRangesList with names c("1", "3"). If there are a total of 0 ORFs, an empty IRangesList will be returned.

**See Also**

Other findORFs: [findMapORFs\(\)](#), [findORFsFasta\(\)](#), [findUORFs\(\)](#), [startDefinition\(\)](#), [stopDefinition\(\)](#)

**Examples**

```
findORFs("ATGTAA")
findORFs("ATGTTAA") # not in frame anymore

findORFs("ATGATGTA") # two ORFs
findORFs("ATGATGTA", longestORF = TRUE) # only longest of two above

findORFs(c("ATGTAA", "ATGATGTA"))
```

---

findORFsFasta

*Finds Open Reading Frames in fasta files.*

---

**Description**

Should be used for procaryote genomes or transcript sequences as fasta. Makes no sense for eukaryote whole genomes, since it contains splicing. Searches through each fasta header and reports all ORFs found for BOTH sense (+) and antisense strand (-) in all frames. Name of the header will be used as seqnames of reported ORFs. Each fasta header is treated separately, and name of the sequence will be used as seqname in returned GRanges object. This supports circular genomes.

**Usage**

```
findORFsFasta(
  filePath,
  startCodon = startDefinition(1),
  stopCodon = stopDefinition(1),
  longestORF = TRUE,
  minimumLength = 0,
  is.circular = FALSE
)
```

**Arguments**

filePath	(character) Path to the fasta file. Can be both uppercase or lowercase.
startCodon	(character vector) Possible START codons to search for. Check <a href="#">startDefinition</a> for helper function.
stopCodon	(character vector) Possible STOP codons to search for. Check <a href="#">stopDefinition</a> for helper function.

longestORF	(logical) Default TRUE. Keep only the longest ORF per unique (seqname, strand, stopcodon) combination, you can also use function <a href="#">longestORFs</a> after creation of ORFs for same result.
minimumLength	(integer) Default is 0. Which is START + STOP = 6 bp. Minimum length of ORF, without counting 3bps for START and STOP codons. For example minimumLength = 8 will result in size of ORFs to be at least START + 8*3 (bp) + STOP = 30 bases. Use this param to restrict search.
is.circular	(logical) Whether the genome in filePath is circular. Prokaryotic genomes are usually circular. Be carefull if you want to extract sequences, remember that seqlengths must be set, else it does not know what last base in sequence is before loop ends!

### Details

Remember if you have a fasta file of transcripts (transcript coordinates), delete all negative stranded ORFs afterwards by: `orfs <- orfs[strandBool(orfs)]` # negative strand orfs make no sense then. Seqnames are created from header by format: `>name info`, so name must be first after "biggern than" and space between name and info. Also make sure your fasta file is valid (no hidden spaces etc), as this might break the coordinate system!

### Value

(GRanges) object of ORFs mapped from fasta file. Positions are relative to the fasta file.

### See Also

Other findORFs: [findMapORFs\(\)](#), [findORFs\(\)](#), [findUORFs\(\)](#), [startDefinition\(\)](#), [stopDefinition\(\)](#)

### Examples

```
# location of the example fasta file
example_genome <- system.file("extdata", "genome.fasta", package = "ORFik")
findORFsFasta(example_genome)
```

---

findPeaksPerGene	<i>Find peaks per gene</i>
------------------	----------------------------

---

### Description

For finding the peaks (stall sites) per gene, with some default filters. A peak is basically a position of very high coverage compared to its surrounding area, as measured using zscore.

### Usage

```
findPeaksPerGene(
  tx,
  reads,
  top_tx = 0.5,
  min_reads_per_tx = 20,
  min_reads_per_peak = 10,
  type = "max"
)
```

**Arguments**

tx	a GRangesList
reads	a GAlignments or GRanges, must be 1 width reads like p-shifts, or other reads that is single positioned.
top_tx	numeric, default 0.50 (only use 50% top transcripts by read counts).
min_reads_per_tx	numeric, default 20. Gene must have at least 20 reads, applied before type filter.
min_reads_per_peak	numeric, default 10. Peak must have at least 10 reads.
type	character, default "max". Get only max peak per gene. Alternatives: "all", all peaks passing the input filter will be returned. "median", only peaks that is higher than the median of all peaks. "maxmedian": get first "max", then median of those.

**Details**

For more details see reference, which uses a slightly different method by zscore of a sliding window instead of over the whole tx.

**Value**

a data.table of gene\_id, position, counts of the peak, zscore and standard deviation of the peak compared to rest of gene area.

**References**

doi: 10.1261/rna.065235.117

**Examples**

```
df <- ORFik.template.experiment()
cds <- loadRegion(df, "cds")
# Load ribo seq from ORFik
rfp <- fimport(df[3,]$filepath)
# All transcripts passing filter
findPeaksPerGene(cds, rfp, top_tx = 0)
# Top 50% of genes
findPeaksPerGene(cds, rfp)
```

---

findUORFs

*Find upstream ORFs from transcript annotation*


---

**Description**

Procedure: 1. Create a new search space starting with the 5' UTRs. 2. Redefine TSS with CAGE if wanted. 3. Add the whole of CDS to search space to allow uORFs going into cds. 4. find ORFs on that search space. 5. Filter out wrongly found uORFs, if CDS is included. The CDS, alternative CDS, uORFs starting within the CDS etc.

**Usage**

```

findUORFs(
  fiveUTRs,
  fa,
  startCodon = startDefinition(1),
  stopCodon = stopDefinition(1),
  longestORF = TRUE,
  minimumLength = 0,
  cds = NULL,
  cage = NULL,
  extension = 1000,
  filterValue = 1,
  restrictUpstreamToTx = FALSE,
  removeUnused = FALSE
)

```

**Arguments**

fiveUTRs	(GRangesList) The 5' leaders or full transcript sequences
fa	a <a href="#">FaFile</a> . With fasta sequences corresponding to fiveUTR annotation. Usually loaded from the genome of an organism with <code>fa = ORFik::findFa("path/to/fasta/genome")</code>
startCodon	(character vector) Possible START codons to search for. Check <a href="#">startDefinition</a> for helper function.
stopCodon	(character vector) Possible STOP codons to search for. Check <a href="#">stopDefinition</a> for helper function.
longestORF	(logical) Default TRUE. Keep only the longest ORF per unique (seqname, strand, stopcodon) combination, you can also use function <a href="#">longestORFs</a> after creation of ORFs for same result.
minimumLength	(integer) Default is 0. Which is START + STOP = 6 bp. Minimum length of ORF, without counting 3bps for START and STOP codons. For example <code>minimumLength = 8</code> will result in size of ORFs to be at least START + 8*3 (bp) + STOP = 30 bases. Use this param to restrict search.
cds	(GRangesList) CDS of relative fiveUTRs, applicable only if you want to extend 5' leaders downstream of CDS's, to allow upstream ORFs that can overlap into CDS's.
cage	Either a filePath for the CageSeq file as .bed .bam or .wig, with possible compressions (".gzip", ".gz", ".bgz"), or already loaded CageSeq peak data as GRanges or GAlignment. NOTE: If it is a .bam file, it will add a score column by running: <code>convertToOneBasedRanges(cage, method = "5prime", addScoreColumn = TRUE)</code> The score column is then number of replicates of read, if score column is something else, like read length, set the score column to NULL first.
extension	The maximum number of bases upstream of the TSS to search for CageSeq peak.
filterValue	The minimum number of reads on cage position, for it to be counted as possible new tss. (represented in score column in CageSeq data) If you already filtered, set it to 0.
restrictUpstreamToTx	a logical (FALSE). If TRUE: restrict leaders to not extend closer than 5 bases from closest upstream leader, set this to TRUE.
removeUnused	logical (FALSE), if False: (standard is to set them to original annotation), If TRUE: remove leaders that did not have any cage support.

**Details**

From default a filtering process is done to remove "fake" uORFs, but only if cds is included, since uORFs that stop on the stop codon on the CDS is not a uORF, but an alternative cds by definition, etc.

**Value**

A GRangesList of uORFs, 1 granges list element per uORF.

**See Also**

Other findORFs: [findMapORFs\(\)](#), [findORFsFasta\(\)](#), [findORFs\(\)](#), [startDefinition\(\)](#), [stopDefinition\(\)](#)

**Examples**

```
# Load annotation
txdbFile <- system.file("extdata", "hg19_knownGene_sample.sqlite",
                        package = "GenomicFeatures")

## Not run:
txdb <- loadTxdb(txdbFile)
fiveUTRs <- loadRegion(txdb, "leaders")
cds <- loadRegion(txdb, "cds")
if (requireNamespace("BSgenome.Hsapiens.UCSC.hg19")) {
  # Normally you would not use a BSgenome, but some custom fasta-
  # annotation you have for your species
  findUORFs(fiveUTRs, BSgenome.Hsapiens.UCSC.hg19::Hsapiens, "ATG",
            cds = cds)
}

## End(Not run)
```

---

<code>firstEndPerGroup</code>	<i>Get first end per granges group</i>
-------------------------------	--

---

**Description**

grl must be sorted, call `ORFik:::sortPerGroup` if needed

**Usage**

```
firstEndPerGroup(grl, keep.names = TRUE)
```

**Arguments**

```
grl          a GRangesList
keep.names   a boolean, keep names or not, default: (TRUE)
```

**Value**

a `Rle(keep.names = T)`, or integer vector(F)

**Examples**

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
  ranges = IRanges(c(7, 14), width = 3),
  strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
  ranges = IRanges(c(4, 1), c(9, 3)),
  strand = c("-", "-"))
grl <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
firstEndPerGroup(grl)
```

---

firstExonPerGroup      *Get first exon per GRangesList group*

---

**Description**

grl must be sorted, call `ORFik:::sortPerGroup` if needed

**Usage**

```
firstExonPerGroup(grl)
```

**Arguments**

grl                    a [GRangesList](#)

**Value**

a `GRangesList` of the first exon per group

**Examples**

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
  ranges = IRanges(c(7, 14), width = 3),
  strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
  ranges = IRanges(c(4, 1), c(9, 3)),
  strand = c("-", "-"))
grl <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
firstExonPerGroup(grl)
```

---

firstStartPerGroup      *Get first start per granges group*

---

**Description**

grl must be sorted, call `ORFik:::sortPerGroup` if needed

**Usage**

```
firstStartPerGroup(grl, keep.names = TRUE)
```

**Arguments**

grl                    a [GRangesList](#)  
 keep.names          a boolean, keep names or not, default: (TRUE)

**Value**

a Rle(keep.names = TRUE), or integer vector(FALSE)

**Examples**

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
                  ranges = IRanges(c(7, 14), width = 3),
                  strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
                   ranges = IRanges(c(4, 1), c(9, 3)),
                   strand = c("-", "-"))
grl <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
firstStartPerGroup(grl)
```

floss

*Fragment Length Organization Similarity Score***Description**

This feature is usually calculated only for RiboSeq reads. For reads of width between ‘start’ and ‘end’, sum the fraction of RiboSeq reads (per read widths) that overlap ORFs and normalize by CDS read width fractions. So if all read length are width 34 in ORFs and CDS, value is 1. If width is 33 in ORFs and 34 in CDS, value is 0. If width is 33 in ORFs and 50/50 (33 and 34) in CDS, values will be 0.5 (for 33).

**Usage**

```
floss(grl, RFP, cds, start = 26, end = 34, weight = 1L)
```

**Arguments**

grl                    a [GRangesList](#) object can be either transcripts, 5’ utrs, cds’, 3’ utrs or ORFs as a special case (uORFs, potential new cds’ etc). If regions are not spliced you can send a [GRanges](#) object.

RFP                    ribosomal footprints, given as [GAlignments](#) or [GRanges](#) object, must be already shifted and resized to the p-site. Requires a \$size column with original read lengths.

cds                    a [GRangesList](#) of coding sequences, cds has to have names as grl so that they can be matched

start                  usually 26, the start of the floss interval (inclusive)

end                    usually 34, the end of the floss interval (inclusive)

weight                a vector (default: 1L, if 1L it is identical to countOverlaps()), if single number (!= 1), it applies for all, if more than one must be equal size of ‘reads’. else it must be the string name of a defined meta column in subject "reads", that gives number of times a read was found. `GRanges("chr1", 1, "+", score = 5)`, would mean "score" column tells that this alignment region was found 5 times.



## Details

Pseudo explanation of the function:

$$\text{SUM}[\text{start to stop}]((\text{grl}[\text{start:end}][\text{name}]/\text{grl}) / (\text{cds}[\text{start:end}][\text{name}]/\text{cds}))$$

Where 'name' is transcript names.

Please read more in the article.

## Value

a vector of FLOSS of length same as grl, 0 means no RFP reads in range, 1 is perfect match.

## References

doi: 10.1016/j.celrep.2014.07.045

## See Also

Other features: [computeFeaturesCage\(\)](#), [computeFeatures\(\)](#), [countOverlapsW\(\)](#), [disengagementScore\(\)](#), [distToCds\(\)](#), [distToTSS\(\)](#), [entropy\(\)](#), [fpkm\\_calc\(\)](#), [fpkm\(\)](#), [fractionLength\(\)](#), [initiationScore\(\)](#), [insideOutsideORF\(\)](#), [isInFrame\(\)](#), [isOverlapping\(\)](#), [kozakSequenceScore\(\)](#), [orfScore\(\)](#), [rankOrder\(\)](#), [ribosomeReleaseScore\(\)](#), [ribosomeStallingScore\(\)](#), [startRegionCoverage\(\)](#), [startRegion\(\)](#), [stopRegion\(\)](#), [subsetCoverage\(\)](#), [translationalEff\(\)](#)

## Examples

```
ORF1 <- GRanges(seqnames = "1",
               ranges = IRanges(start = c(1, 12, 22),
                               end = c(10, 20, 32)),
               strand = "+")
grl <- GRangesList(tx1_1 = ORF1)
# RFP is 1 width position based GRanges
RFP <- GRanges("1", IRanges(c(1, 25, 35, 38), width = 1), "+")
RFP$size <- c(28, 28, 28, 29) # original width in size col
cds <- GRangesList(tx1 = GRanges("1", IRanges(35, 44), "+"))
# grl must have same names as cds + _1 etc, so that they can be matched.
floss(grl, RFP, cds)
# or change ribosome start/stop, more strict
floss(grl, RFP, cds, 28, 28)

# With repeated alignments in score column
ORF2 <- GRanges(seqnames = "1",
               ranges = IRanges(start = c(12, 22, 36),
                               end = c(20, 32, 38)),
               strand = "+")
grl <- GRangesList(tx1_1 = ORF1, tx1_2 = ORF2)
score(RFP) <- c(5, 10, 5, 10)
floss(grl, RFP, cds, weight = "score")
```

---

footprints.analysis    *Pre shifting plot analysis*

---

### Description

For internal use only!

### Usage

```
footprints.analysis(rw, heatmap, region = "start of CDS")
```

### Arguments

rw	a data.table of position, score and fraction (read length) of either TIS or TES (translation end site, around 3' UTR)
heatmap	a logical or character string, default FALSE. If TRUE, will plot heatmap of raw reads before p-shifting to console, to see if shifts given make sense. You can also set a filepath to save the file there.
region	a character string, default "start of CDS"

### Value

invisible(NULL)

---

fpkm                      *Create normalizations of overlapping read counts.*

---

### Description

FPKM is short for "Fragments Per Kilobase of transcript per Million fragments in library". When calculating RiboSeq data FPKM over ORFs, use ORFs as 'grl'. When calculating RNASeq data FPKM, use full transcripts as 'grl'. It is equal to RPKM given that you do not have paired end reads.

### Usage

```
fpkm(grl, reads, pseudoCount = 0, librarySize = "full", weight = 1L)
```

### Arguments

grl	a <a href="#">GRangesList</a> object can be either transcripts, 5' utrs, cds', 3' utrs or ORFs as a special case (uORFs, potential new cds' etc). If regions are not spliced you can send a <a href="#">GRanges</a> object.
reads	a <a href="#">GAlignments</a> , <a href="#">GRanges</a> or <a href="#">GRangesList</a> object, usually of RiboSeq, RnaSeq, CageSeq, etc.
pseudoCount	an integer, by default is 0, set it to 1 if you want to avoid NA and inf values.

librarySize	either numeric value or character vector. Default ("full"), number of alignments in library (reads). If you just have a subset, you can give the value by librarySize = length(wholeLib), if you want lib size to be only number of reads overlapping grl, do: librarySize = "overlapping" sum(countOverlaps(reads, grl) > 0), if reads[1] has 3 hits in grl, and reads[2] has 2 hits, librarySize will be 2, not 5. You can also get the inverse overlap, if you want lib size to be total number of overlaps, do: librarySize = "DESeq" This is standard fpkm way of DESeq2::fpkm(robust = FALSE) sum(countOverlaps(grl, reads)) if grl[1] has 3 reads and grl[2] has 2 reads, librarySize is 5, not 2.
weight	a vector (default: 1L, if 1L it is identical to countOverlaps()), if single number (!= 1), it applies for all, if more than one must be equal size of 'reads'. else it must be the string name of a defined meta column in subject "reads", that gives number of times a read was found. GRanges("chr1", 1, "+", score = 5), would mean "score" column tells that this alignment region was found 5 times.

### Details

Note also that you must consider if you will use the whole read library or just the reads overlapping 'grl' for library size. A normal question here is, does it make sense to include rRNA in library size ? If you only want overlapping grl, do: librarySize = "overlapping"

### Value

a numeric vector with the fpkm values

### References

doi: 10.1038/nbt.1621

### See Also

Other features: [computeFeaturesCage\(\)](#), [computeFeatures\(\)](#), [countOverlapsW\(\)](#), [disengagementScore\(\)](#), [distToCds\(\)](#), [distToTSS\(\)](#), [entropy\(\)](#), [floss\(\)](#), [fpkm\\_calc\(\)](#), [fractionLength\(\)](#), [initiationScore\(\)](#), [insideOutsideORF\(\)](#), [isInFrame\(\)](#), [isOverlapping\(\)](#), [kozakSequenceScore\(\)](#), [orfScore\(\)](#), [rankOrder\(\)](#), [ribosomeReleaseScore\(\)](#), [ribosomeStallingScore\(\)](#), [startRegionCoverage\(\)](#), [startRegion\(\)](#), [stopRegion\(\)](#), [subsetCoverage\(\)](#), [translationalEff\(\)](#)

### Examples

```
ORF <- GRanges(seqnames = "1",
               ranges = IRanges(start = c(1, 10, 20),
                               end = c(5, 15, 25)),
               strand = "+")
grl <- GRangesList(tx1_1 = ORF)
RFP <- GRanges("1", IRanges(25, 25), "+")
fpkm(grl, RFP)

# With weights (10 reads at position 25)
RFP <- GRanges("1", IRanges(25, 25), "+", score = 10)
fpkm(grl, RFP, weight = "score")
```

---

fpkm_calc	<i>Create normalizations of read counts</i>
-----------	---

---

**Description**

A helper for [fpkm()] Normally use function [fpkm()], if you want unusual normalization , you can use this. Short for: Fragments per kilobase of transcript per million fragments Normally used in Translations efficiency calculations

**Usage**

```
fpkm_calc(counts, lengthSize, librarySize)
```

**Arguments**

counts	a list, # of read hits per group
lengthSize	a list of lengths per group
librarySize	a numeric of size 1, the # of reads in library

**Value**

a numeric vector

**References**

doi: 10.1038/nbt.1621

**See Also**

Other features: [computeFeaturesCage\(\)](#), [computeFeatures\(\)](#), [countOverlapsW\(\)](#), [disengagementScore\(\)](#), [distToCds\(\)](#), [distToTSS\(\)](#), [entropy\(\)](#), [floss\(\)](#), [fpkm\(\)](#), [fractionLength\(\)](#), [initiationScore\(\)](#), [insideOutsideORF\(\)](#), [isInFrame\(\)](#), [isOverlapping\(\)](#), [kozakSequenceScore\(\)](#), [orfScore\(\)](#), [rankOrder\(\)](#), [ribosomeReleaseScore\(\)](#), [ribosomeStallingScore\(\)](#), [startRegionCoverage\(\)](#), [startRegion\(\)](#), [stopRegion\(\)](#), [subsetCoverage\(\)](#), [translationalEff\(\)](#)

---

fractionLength	<i>Fraction Length</i>
----------------	------------------------

---

**Description**

Fraction Length is defined as

$$(\text{widths of grl})/\text{tx\_len}$$

so that each group in the grl is divided by the corresponding transcript.

**Usage**

```
fractionLength(grl, tx_len)
```

**Arguments**

- `grl` a [GRangesList](#) object with usually either leaders, cds', 3' utrs or ORFs. ORFs are a special case, see argument `tx_len`
- `tx_len` the transcript lengths of the transcripts, a named (tx names) vector of integers. If you have the transcripts as [GRangesList](#), call `'ORFik:::widthPerGroup(tx, TRUE)'`.  
If you used `CageSeq` to reannotate leaders, then the tss for the the leaders have changed, therefore the tx lengths have changed. To account for that call: `'tx_len <- widthPerGroup(extendLeaders(tx, cageFiveUTRs))'` and calculate fraction length using `'fractionLength(grl, tx_len)'`.

**Value**

a numeric vector of ratios

**References**

doi: 10.1242/dev.098343

**See Also**

Other features: [computeFeaturesCage\(\)](#), [computeFeatures\(\)](#), [countOverlapsW\(\)](#), [disengagementScore\(\)](#), [distToCds\(\)](#), [distToTSS\(\)](#), [entropy\(\)](#), [floss\(\)](#), [fpkm\\_calc\(\)](#), [fpkm\(\)](#), [initiationScore\(\)](#), [insideOutsideORF\(\)](#), [isInFrame\(\)](#), [isOverlapping\(\)](#), [kozakSequenceScore\(\)](#), [orfScore\(\)](#), [rankOrder\(\)](#), [ribosomeReleaseScore\(\)](#), [ribosomeStallingScore\(\)](#), [startRegionCoverage\(\)](#), [startRegion\(\)](#), [stopRegion\(\)](#), [subsetCoverage\(\)](#), [translationalEff\(\)](#)

**Examples**

```
ORF <- GRanges(seqnames = "1",
               ranges = IRanges(start = c(1, 10, 20), end = c(5, 15, 25)),
               strand = "+")
grl <- GRangesList(tx1_1 = ORF)
# grl must have same names as cds + _1 etc, so that they can be matched.
tx <- GRangesList(tx1 = GRanges("1", IRanges(1, 50), "+"))
fractionLength(grl, ORFik:::widthPerGroup(tx, keep.names = TRUE))
```

---

fread.bed

*Load bed file as GRanges*

---

**Description**

Wraps around [import.bed](#) and tries to speed up loading with the use of `data.table`. Supports `gzip`, `gz`, `bgz` and `bed` formats. Also safer chromosome naming with the argument `chrStyle`

**Usage**

```
fread.bed(filePath, chrStyle = NULL)
```

**Arguments**

filePath	The location of the bed file
chrStyle	a GRanges object, TxDb, FaFile, or a <a href="#">seqlevelsStyle</a> (Default: NULL) to get seqlevelsStyle from. Is chromosome 1 called chr1 or 1, is mitochondrial chromosome called MT or chrM etc. Will use 1st seqlevel-style if more are present. Like: c("NCBI", "UCSC") -> pick "NCBI"

**Value**

a [GRanges](#) object

**See Also**

Other utils: [bedToGR\(\)](#), [convertToOneBasedRanges\(\)](#), [export.bed12\(\)](#), [export.wiggle\(\)](#), [fimport\(\)](#), [findFa\(\)](#), [optimizeReads\(\)](#), [readBam\(\)](#), [readWig\(\)](#)

**Examples**

```
# path to example CageSeq data from hg19 heart sample
cageData <- system.file("extdata", "cage-seq-heart.bed.bgz",
                        package = "ORFik")
fread.bed(cageData)
```

---

gcContent

*Get GC content*


---

**Description**

0.5 means 50

**Usage**

```
gcContent(seqs, fa = NULL)
```

**Arguments**

seqs	a character vector of sequences, or ranges as GRangesList
fa	fasta index file .fai file, either path to it, or the loaded FaFile, default (NULL), only set if you give ranges as GRangesList

**Value**

a numeric vector of gc content scores

**Examples**

```
# Here we make an example from scratch
seqName <- "Chromosome"
ORF1 <- GRanges(seqnames = seqName,
                ranges = IRanges(c(1007, 1096), width = 60),
                strand = c("+", "+"))
ORF2 <- GRanges(seqnames = seqName,
                ranges = IRanges(c(400, 100), width = 30),
                strand = c("-", "-"))
ORFs <- GRangesList(tx1 = ORF1, tx2 = ORF2)
# get path to FaFile for sequences
faFile <- system.file("extdata", "genome.fasta", package = "ORFik")
gcContent(ORFs, faFile)
```

---

getGAlignments

*Internal GAlignments loader from fst data.frame*


---

**Description**

Internal GAlignments loader from fst data.frame

**Usage**

```
getGAlignments(df)
```

**Arguments**

df a data.frame with columns minimum 4 columns: seqnames, start ("pos" in final GA object), strand and width.  
Additional columns will be assigned as meta columns

**Value**

GAlignments object

---

getGAlignmentsPairs

*Internal GAlignmentPairs loader from fst data.frame*


---

**Description**

Internal GAlignmentPairs loader from fst data.frame

**Usage**

```
getGAlignmentsPairs(df)
```

**Arguments**

df a data.frame with columns minimum 6 columns: seqnames, start1/start2 (integers), cigar1/cigar2 and strand  
Additional columns will be assigned as meta columns

**Value**

GAlignmentPairs object

---

getGenomeAndAnnotation

*Download genome (fasta), annotation (GTF) and contaminants*

---

**Description**

This function automatically downloads (if files not already exists) genomes and contaminants specified for genome alignment. Will create a R transcript database (TxDb object) from the annotation. It will also index the genome for you  
 If you misspelled something or crashed, delete wrong files and run again.  
 Do remake = TRUE, to do it all over again.

**Usage**

```
getGenomeAndAnnotation(
  organism,
  output.dir,
  db = "ensembl",
  GTF = TRUE,
  genome = TRUE,
  merge_contaminants = TRUE,
  phix = FALSE,
  ncRNA = FALSE,
  tRNA = FALSE,
  rRNA = FALSE,
  gunzip = TRUE,
  remake = FALSE,
  assembly_type = "primary_assembly"
)
```

**Arguments**

organism	scientific name of organism, Homo sapiens, Danio rerio, Mus musculus, etc.
output.dir	directory to save downloaded data
db	database to use for genome and GTF, default advised: "ensembl" (will contain haplotypes, large file!). Alternatives: "refseq" (primary assembly) and "genbank" (mix)
GTF	logical, default: TRUE, download gtf of organism specified in "organism" argument. If FALSE, check if the downloaded file already exist. If you want to use a custom gtf from you hard drive, set GTF = FALSE, and assign: annotation <- getGenomeAndAnnotation(gtf = FALSE) annotation["gtf"] = "path/to/gtf.gtf". Only db = "ensembl" allowed for GTF.
genome	logical, default: TRUE, download genome of organism specified in "organism" argument. If FALSE, check if the downloaded file already exist. If you want to use a custom gtf from you hard drive, set GTF = FALSE, and assign:



```

annotation <- getGenomeAndAnnotation(genome = FALSE)
annotation["genome"] = "path/to/genome.fasta".
Will download the primary assembly for ensembl

```

merge_contaminants	logical, default TRUE. Will merge the contaminants specified into one fasta file, this considerably saves space and is much quicker to align with STAR than each contaminant on its own. If no contaminants are specified, this is ignored.
phix	logical, default FALSE, download phix sequence to filter out with. Phix is used as a contaminant genome. Only use if illumina sequencing. Phix is used in Illumina sequencers for sequencing quality control. Genome is: refseq, Escherichia virus phiX174
ncRNA	logical or character, default FALSE (not used, no download), ncRNA is used as a contaminant genome. If TRUE, will try to find ncRNA sequences from the gtf file, usually represented as lncRNA (long noncoding RNA's). Will let you know if no ncRNA sequences were found in gtf. If not found try character input: Alternatives: "auto" or manual assign like "human". If "auto" will try to find ncRNA file on NONCODE from organism, Homo sapiens -> human etc. "auto" will not work for all, then you must specify the name used by NONCODE, go to the link below and find it. If not "auto" / "" it must be a character vector of species common name (not scientific name) Homo sapiens is human, Rattus norvegicus is rat etc, download ncRNA sequence to filter out with. From NONCODE online server, if you cant find common name see: <a href="http://www.noncode.org/download.php/">http://www.noncode.org/download.php/</a>
tRNA	logical or character, default FALSE (not used, no download), tRNA is used as a contaminant genome. If TRUE, will try to find tRNA sequences from the gtf file, usually represented as Mt_tRNA (mature tRNA's). Will let you know if no tRNA sequences were found in gtf. If not found try character input: if not "" it must be a character vector to valid path of mature tRNAs fasta file to remove as contaminants on your disc. Find and download your wanted mtRNA at: <a href="http://gtrnadb.ucsc.edu/">http://gtrnadb.ucsc.edu/</a> , or run trna-scan on you genome.
rRNA	logical or character, default FALSE (not used, no download), rRNA is used as a contaminant genome. If TRUE, will try to find rRNA sequences from the gtf file, usually represented as rRNA (ribosomal RNA's). Will let you know if no rRNA sequences were found in gtf. If not found you can try character input: If "silva" will download silva SSU & LSU sequences for all species (250MB file) and use that. If you want a smaller file go to <a href="https://www.arb-silva.de/">https://www.arb-silva.de/</a> If not "" or "silva" it must be a character vector to valid path of mature rRNA fasta file to remove as contaminants on your disc.
gunzip	logical, default TRUE, uncompress downloaded files that are zipped when downloaded, should be TRUE!
remake	logical, default: FALSE, if TRUE remake everything specified
assembly_type	a character string specifying from which assembly type the genome shall be retrieved from (ensembl only, else this argument is ignored): Default is assembly_type = "primary_assembly"). This will give you all no copies of any chromosomes. As an example, the primary_assembly fasta genome in human is only a few GB uncompressed. assembly_type = "toplevel"). This will give you all multi-chromosomes (copies of the same chromosome with small variations). As an example the toplevel fasta genome in human is over 70 GB uncompressed. To get primary assembly with 1 chromosome variant per chromosome:

**Details**

If you want custom genome or gtf from you hard drive, assign it after you run this function, like this:

```
annotation <- getGenomeAndAnnotation(GTF = FALSE, genome = FALSE)
annotation["genome"] = "path/to/genome.fasta"
annotation["gtf"] = "path/to/gtf.gtf"
```

**Value**

a named character vector of path to genomes and gtf downloaded, and additional contaminants if used. If merge\_contaminants is TRUE, will not give individual fasta files to contaminants, but only the merged one.

**See Also**

Other STAR: [STAR.align.folder\(\)](#), [STAR.align.single\(\)](#), [STAR.allsteps.multiQC\(\)](#), [STAR.index\(\)](#), [STAR.install\(\)](#), [STAR.multiQC\(\)](#), [STAR.remove.crashed.genome\(\)](#), [install.fastp\(\)](#)

**Examples**

```
output.dir <- "/Bio_data/references/zebrafish"
#getGenomeAndAnnotation("Danio rerio", output.dir)

## Get Phix contamints to deplete during alignment
#getGenomeAndAnnotation("Danio rerio", output.dir, phix = TRUE)
```

---

getGRanges

*Internal GRanges loader from fst data.frame*


---

**Description**

Internal GRanges loader from fst data.frame

**Usage**

```
getGRanges(df)
```

**Arguments**

df a data.frame with columns minimum 4 columns: seqnames, start, strand and width.  
Additional columns will be assigned as meta columns

**Value**

GRanges object

---

getNGenesCoverage	<i>Get number of genes per coverage table</i>
-------------------	---

---

**Description**

Used to count genes in ORFik meta plots

**Usage**

```
getNGenesCoverage(coverage)
```

**Arguments**

coverage	a data.table with coverage
----------	----------------------------

**Value**

number of genes in coverage

---

getWeights	<i>Get weights from a subject GenomicRanges object</i>
------------	--

---

**Description**

Get weights from a subject GenomicRanges object

**Usage**

```
getWeights(subject, weight = 1L)
```

**Arguments**

subject	a GRanges, IRanges or GAlignment object
weight	a vector (default: 1L, if 1L it is identical to countOverlaps()), if single number (!= 1), it applies for all, if more than one must be equal size of 'reads'. else it must be the string name of a defined meta column in subject "reads", that gives number of times a read was found. GRanges("chr1", 1, "+", score = 5), would mean "score" column tells that this alignment region was found 5 times.

**Value**

a numeric vector of weights of equal size to subject

---

get\_genome\_fasta      *Download genome (fasta), annotation (GTF) and contaminants*

---

### Description

This function automatically downloads (if files not already exists) genomes and contaminants specified for genome alignment. Will create a R transcript database (TxDb object) from the annotation. It will also index the genome for you

If you misspelled something or crashed, delete wrong files and run again.

Do remake = TRUE, to do it all over again.

### Usage

```
get_genome_fasta(genome, output.dir, organism, assembly_type, db, gunzip)
```

### Arguments

genome	logical, default: TRUE, download genome of organism specified in "organism" argument. If FALSE, check if the downloaded file already exist. If you want to use a custom gtf from you hard drive, set GTF = FALSE, and assign: <code>annotation &lt;- getGenomeAndAnnotation(genome = FALSE)</code> <code>annotation["genome"] = "path/to/genome.fasta"</code> Will download the primary assembly for ensembl
output.dir	directory to save downloaded data
organism	scientific name of organism, Homo sapiens, Danio rerio, Mus musculus, etc.
assembly_type	a character string specifying from which assembly type the genome shall be retrieved from (ensembl only, else this argument is ignored): Default is <code>assembly_type = "primary_assembly"</code> ). This will give you all no copies of any chromosomes. As an example, the primary_assembly fasta genome in human is only a few GB uncompressed. <code>assembly_type = "toplevel"</code> ). This will give you all multi-chromosomes (copies of the same chromosome with small variations). As an example the <code>toplevel</code> fasta genome in human is over 70 GB uncompressed. To get primary assembly with 1 chromosome variant per chromosome:
db	database to use for genome and GTF, default advised: "ensembl" (will contain haplotypes, large file!). Alternatives: "refseq" (primary assembly) and "genbank" (mix)
gunzip	logical, default TRUE, uncompress downloaded files that are zipped when downloaded, should be TRUE!

### Details

If you want custom genome or gtf from you hard drive, assign it after you run this function, like this:

```
annotation <- getGenomeAndAnnotation(GTF = FALSE, genome = FALSE)
annotation["genome"] = "path/to/genome.fasta"
annotation["gtf"] = "path/to/gtf.gtf"
```

**Value**

a named character vector of path to genomes and gtf downloaded, and additional contaminants if used. If merge\_contaminants is TRUE, will not give individual fasta files to contaminants, but only the merged one.

**See Also**

Other STAR: `STAR.align.folder()`, `STAR.align.single()`, `STAR.allsteps.multiQC()`, `STAR.index()`, `STAR.install()`, `STAR.multiQC()`, `STAR.remove.crashed.genome()`, `install.fastp()`

**Examples**

```
output.dir <- "/Bio_data/references/zebrafish"
#getGenomeAndAnnotation("Danio rerio", output.dir)

## Get Phix contamints to deplete during alignment
#getGenomeAndAnnotation("Danio rerio", output.dir, phix = TRUE)
```

---

get\_genome\_gtf

*Download genome (fasta), annotation (GTF) and contaminants*


---

**Description**

This function automatically downloads (if files not already exists) genomes and contaminants specified for genome alignment. Will create a R transcript database (TxDb object) from the annotation. It will also index the genome for you  
If you misspelled something or crashed, delete wrong files and run again.  
Do remake = TRUE, to do it all over again.

**Usage**

```
get_genome_gtf(GTF, output.dir, organism, assembly_type, gunzip, genome)
```

**Arguments**

GTF	logical, default: TRUE, download gtf of organism specified in "organism" argument. If FALSE, check if the downloaded file already exist. If you want to use a custom gtf from you hard drive, set GTF = FALSE, and assign: <pre>annotation &lt;- getGenomeAndAnnotation(gtf = FALSE) annotation["gtf"] = "path/to/gtf.gtf".</pre> Only db = "ensembl" allowed for GTF.
output.dir	directory to save downloaded data
organism	scientific name of organism, Homo sapiens, Danio rerio, Mus musculus, etc.
assembly_type	a character string specifying from which assembly type the genome shall be retrieved from (ensembl only, else this argument is ignored): Default is assembly_type = "primary_assembly". This will give you all no copies of any chromosomes. As an example, the primary_assembly fasta genome in human is only a few GB uncompressed. assembly_type = "toplevel"). This will give you all multi-chromosomes (copies of the same chromosome with small variations). As an example the

	<p>toplevel fasta genome in human is over 70 GB uncompressed. To get primary assembly with 1 chromosome variant per chromosome:</p>
gunzip	logical, default TRUE, uncompress downloaded files that are zipped when downloaded, should be TRUE!
genome	a character path, default NULL. if set, must be path to genome fasta file, must be indexed. If you want to make sure chromosome naming of the GTF matches the genome. Not necessary if you downloaded from same source. If value is NULL or FALSE, will be ignored.

### Details

If you want custom genome or gtf from you hard drive, assign it after you run this function, like this:

```
annotation <- getGenomeAndAnnotation(GTF = FALSE, genome = FALSE)
annotation["genome"] = "path/to/genome.fasta"
annotation["gtf"] = "path/to/gtf.gtf"
```

### Value

a named character vector of path to genomes and gtf downloaded, and additional contaminants if used. If merge\_contaminants is TRUE, will not give individual fasta files to contaminants, but only the merged one.

### See Also

Other STAR: [STAR.align.folder\(\)](#), [STAR.align.single\(\)](#), [STAR.allsteps.multiQC\(\)](#), [STAR.index\(\)](#), [STAR.install\(\)](#), [STAR.multiQC\(\)](#), [STAR.remove.crashed.genome\(\)](#), [install.fastp\(\)](#)

### Examples

```
output.dir <- "/Bio_data/references/zebrafish"
#getGenomeAndAnnotation("Danio rerio", output.dir)

## Get Phix contamints to deplete during alignment
#getGenomeAndAnnotation("Danio rerio", output.dir, phix = TRUE)
```

---

get\_noncoding\_rna      *Download genome (fasta), annotation (GTF) and contaminants*

---

### Description

This function automatically downloads (if files not already exists) genomes and contaminants specified for genome alignment. Will create a R transcript database (TxDb object) from the annotation. It will also index the genome for you  
 If you misspelled something or crashed, delete wrong files and run again.  
 Do remake = TRUE, to do it all over again.

### Usage

```
get_noncoding_rna(ncRNA, output.dir, organism, gunzip)
```

**Arguments**

ncRNA	logical or character, default FALSE (not used, no download), ncRNA is used as a contaminant genome. If TRUE, will try to find ncRNA sequences from the gtf file, usually represented as lncRNA (long noncoding RNA's). Will let you know if no ncRNA sequences were found in gtf. If not found try character input: Alternatives: "auto" or manual assign like "human". If "auto" will try to find ncRNA file on NONCODE from organism, Homo sapiens -> human etc. "auto" will not work for all, then you must specify the name used by NONCODE, go to the link below and find it. If not "auto" / "" it must be a character vector of species common name (not scientific name) Homo sapiens is human, Rattus norvegicus is rat etc, download ncRNA sequence to filter out with. From NONCODE online server, if you cant find common name see: <a href="http://www.noncode.org/download.php/">http://www.noncode.org/download.php/</a>
output.dir	directory to save downloaded data
organism	scientific name of organism, Homo sapiens, Danio rerio, Mus musculus, etc.
gunzip	logical, default TRUE, uncompress downloaded files that are zipped when downloaded, should be TRUE!

**Details**

If you want custom genome or gtf from you hard drive, assign it after you run this function, like this:

```
annotation <- getGenomeAndAnnotation(GTF = FALSE, genome = FALSE)
annotation["genome"] = "path/to/genome.fasta"
annotation["gtf"] = "path/to/gtf.gtf"
```

**Value**

a named character vector of path to genomes and gtf downloaded, and additional contaminants if used. If merge\_contaminants is TRUE, will not give individual fasta files to contaminants, but only the merged one.

**See Also**

Other STAR: [STAR.align.folder\(\)](#), [STAR.align.single\(\)](#), [STAR.allsteps.multiQC\(\)](#), [STAR.index\(\)](#), [STAR.install\(\)](#), [STAR.multiQC\(\)](#), [STAR.remove.crashed.genome\(\)](#), [install.fastp\(\)](#)

**Examples**

```
output.dir <- "/Bio_data/references/zebrafish"
#getGenomeAndAnnotation("Danio rerio", output.dir)

## Get Phix contamints to deplete during alignment
#getGenomeAndAnnotation("Danio rerio", output.dir, phix = TRUE)
```

---

get\_phix\_genome      *Download genome (fasta), annotation (GTF) and contaminants*

---

### Description

This function automatically downloads (if files not already exists) genomes and contaminants specified for genome alignment. Will create a R transcript database (TxDb object) from the annotation. It will also index the genome for you  
 If you misspelled something or crashed, delete wrong files and run again.  
 Do remake = TRUE, to do it all over again.

### Usage

```
get_phix_genome(phix, output.dir, gunzip)
```

### Arguments

phix	logical, default FALSE, download phix sequence to filter out with. Phix is used as a contaminant genome. Only use if illumina sequencing. Phix is used in Illumina sequencers for sequencing quality control. Genome is: refseq, Escherichia virus phiX174
output.dir	directory to save downloaded data
gunzip	logical, default TRUE, uncompress downloaded files that are zipped when downloaded, should be TRUE!

### Details

If you want custom genome or gtf from you hard drive, assign it after you run this function, like this:

```
annotation <- getGenomeAndAnnotation(GTF = FALSE, genome = FALSE)
annotation["genome"] = "path/to/genome.fasta"
annotation["gtf"] = "path/to/gtf.gtf"
```

### Value

a named character vector of path to genomes and gtf downloaded, and additional contaminants if used. If merge\_contaminants is TRUE, will not give individual fasta files to contaminants, but only the merged one.

### See Also

Other STAR: [STAR.align.folder\(\)](#), [STAR.align.single\(\)](#), [STAR.allsteps.multiQC\(\)](#), [STAR.index\(\)](#), [STAR.install\(\)](#), [STAR.multiQC\(\)](#), [STAR.remove.crashed.genome\(\)](#), [install.fastp\(\)](#)

### Examples

```
output.dir <- "/Bio_data/references/zebrafish"
#getGenomeAndAnnotation("Danio rerio", output.dir)

## Get Phix contamints to deplete during alignment
#getGenomeAndAnnotation("Danio rerio", output.dir, phix = TRUE)
```



---

get_silva_rRNA	<i>Download Silva SSU &amp; LSU sequences</i>
----------------	---

---

**Description**

Version downloaded is 138.1. NR99\_tax (non redundant)

**Usage**

```
get_silva_rRNA(output.dir)
```

**Arguments**

output.dir      directory to save downloaded data

**Details**

If it fails from timeout, set higher timeout: options(timeout = 200)

**Value**

filepath to downloaded file

**Examples**

```
output.dir <- tempdir()
# get_silva_rRNA(output.dir)
```

---

groupGRangesBy	<i>Group GRanges</i>
----------------	----------------------

---

**Description**

It will group / split the GRanges object by the argument 'other'. For example if you would like to group GRanges object by gene, set other to gene names.

If 'other' is not specified function will try to use the names of the GRanges object. It will then be similar to 'split(gr, names(gr))'.

**Usage**

```
groupGRangesBy(gr, other = NULL)
```

**Arguments**

gr                      a GRanges object  
other                    a vector of unique names to group by (default: NULL)

**Details**

It is important that all intended groups in 'other' are uniquely named, otherwise duplicated group names will be grouped together.

**Value**

a GRangesList named after names(Granges) if other is NULL, else names are from unique(other)

**Examples**

```
ORFranges <- GRanges(seqnames = Rle(rep("1", 3)),
                    ranges = IRanges(start = c(1, 10, 20),
                                     end = c(5, 15, 25)),
                    strand = "+")
ORFranges2 <- GRanges("1",
                    ranges = IRanges(start = c(20, 30, 40),
                                     end = c(25, 35, 45)),
                    strand = "+")
names(ORFranges) = rep("tx1_1", 3)
names(ORFranges2) = rep("tx1_2", 3)
grl <- GRangesList(tx1_1 = ORFranges, tx1_2 = ORFranges2)
gr <- unlist(grl, use.names = FALSE)
## now recreate the grl
## group by orf
grltest <- groupGRangesBy(gr) # using the names to group
identical(grl, grltest) ## they are identical

## group by transcript
names(gr) <- txNames(gr)
grltest <- groupGRangesBy(gr)
identical(grl, grltest) ## they are not identical
```

---

groupings

*Get number of ranges per group as an iteration*

---

**Description**

Get number of ranges per group as an iteration

**Usage**

```
groupings(grl)
```

**Arguments**

```
grl          GRangesList
```

**Value**

an integer vector

**Examples**

```
grl <- GRangesList(GRanges("1", c(1, 3, 5), "+"),
                  GRanges("1", c(19, 21, 23), "+"))
ORFik:::groupings(grl)
```

---

gSort	<i>Sort a GRangesList, helper.</i>
-------	------------------------------------

---

**Description**

A helper for [sortPerGroup()]. A faster, more versatile reimplementaion of GenomicRanges::sort() Normally not used directly. Groups first each group, then either decreasing or increasing (on starts if byStarts == T, on ends if byStarts == F)

**Usage**

```
gSort(grl, decreasing = FALSE, byStarts = TRUE)
```

**Arguments**

grl	a <a href="#">GRangesList</a>
decreasing	should the first in each group have max(start(group)) ->T or min-> default(F) ?
byStarts	a logical T, should it order by starts or ends F.

**Value**

an equally named GRangesList, where each group is sorted within group.

---

hasHits	<i>Hits from reads</i>
---------	------------------------

---

**Description**

Finding GRanges groups that have overlap hits with reads Similar to

**Usage**

```
hasHits(grl, reads, keep.names = FALSE, overlaps = NULL)
```

**Arguments**

grl	a <a href="#">GRangesList</a> or GRanges object
reads	a GRanges, GAlignment or GAlignmentPairs object
keep.names	logical (F), keep names or not
overlaps	default NULL, if not null must be countOverlaps(grl, reads), input if you have it already.

**Value**

a list of logicals, T == hit, F == no hit

heatMapL

*Coverage heatmap of multiple libraries***Description**

Coverage heatmap of multiple libraries

**Usage**

```
heatMapL(
  region,
  tx,
  df,
  outdir,
  scores = "sum",
  upstream,
  downstream,
  zeroPosition = upstream,
  acceptedLengths = NULL,
  type = "ofst",
  legendPos = "right",
  colors = "default",
  addFracPlot = TRUE,
  location = "TIS",
  shifting = NULL,
  skip.last = FALSE,
  format = ".png",
  plot.together = TRUE,
  title = TRUE
)
```

**Arguments**

region	#' a <a href="#">GRangesList</a> object of region, usually either leaders, cds', 3' utrs or ORFs, start region, stop regions etc. This is the region that will be mapped in heatmap
tx	default NULL, a <a href="#">GRangesList</a> of transcripts or (container region), names of tx must contain all grl names. The names of grl can also be the ORFik orf names. that is "txName_id"
df	an ORFik <a href="#">experiment</a>
outdir	a character path to directory to save plot, will be named from ORFik experiment columns
scores	character vector, default c("transcriptNormalized", "sum"), either of zscore, transcriptNormalized, sum, mean, median, .. see ?coverageScorings for info and more alternatives.
upstream	1 or 2 integers, default c(50, 30), how long upstream from 0 should window extend (first index is 5' end extension, second is 3' end extension). If only 1 shifting, only 1 value should be given, if two are given will use first.
downstream	1 or 2 integers, default c(29, 69), how long upstream from 0 should window extend (first index is 5' end extension, second is 3' end extension). If only 1 shifting, only 1 value should be given, if two are given will use first.

zeroPosition	an integer DEFAULT (upstream), what is the center point? Like leaders and cds combination, then 0 is the TIS and -1 is last base in leader. NOTE!: if windows have different widths, this will be ignored.
acceptedLengths	an integer vector (NULL), the read lengths accepted. Default NULL, means all lengths accepted.
type	character, default: "ofst". Type of library: either "default", usually bam format (the one you gave to experiment), "pshifted" pshifted reads, "ofst", "bed", "bedo" optimized bed, or "wig"
legendPos	a character, Default "right". Where should the fill legend be ? ("top", "bottom", "right", "left")
colors	character vector, default: "default", this gives you: c("white", "yellow2", "yellow3", "lightblue", "blue", "navy"), do "high" for more high contrasts, or specify your own colors.
addFracPlot	Add margin histogram plot on top of heatmap with fractions per positions
location	a character, default "start site", will make xlabel of heatmap be Position relative to "start site" or alternative given.
shifting	a character, default c("5prime", "3prime"), can also be either or NULL (no shifting of reads)
skip.last	skip top(highest) read length, default FALSE
format	a character, default ".png", alternative ".pdf"
plot.together	logical (default: FALSE), plot all in 1 plot (if TRUE)
title	a character, default NULL (no title), what is the top title of plot?

**Value**

invisible(NULL), plots are saved

**See Also**

Other heatmaps: [coverageHeatMap\(\)](#), [heatMapRegion\(\)](#), [heatMap\\_single\(\)](#)

---

heatMapRegion	<i>Create coverage heatmaps of specified region</i>
---------------	---

---

**Description**

Simplified input space for easier abstraction of coverage heatmaps  
 Pick your region and plot  
 Input CAGE file if you use TSS and want improved 5' annotation.

**Usage**

```
heatMapRegion(
  df,
  region = "TIS",
  outdir = "default",
  scores = c("transcriptNormalized", "sum"),
```

```

type = "ofst",
cage = NULL,
format = ".png",
acceptedLengths = 21:75,
upstream = c(50, 30),
downstream = c(29, 69),
shifting = c("5prime", "3prime")
)

```

### Arguments

df	an ORFik <a href="#">experiment</a>
region	a character, default "TIS", can be any combination of the set: c("TSS", "TIS", "TTS"), which are: Transcription start site (5' end of mrna), Translation initiation site (5' end of CDS), Translation termination site (3' end of CDS)
outdir	a character path, default: "default", saves to: <code>paste0(dirname(df\$filepath[1]), "/QC_STATS/heatMapRegion")</code> , a created folder within the ORFik experiment data folder for plots. Change if you want custom location.
scores	character vector, default c("transcriptNormalized", "sum"), either of zscore, transcriptNormalized, sum, mean, median, .. see <code>?coverageScorings</code> for info and more alternatives.
type	character, default: "ofst". Type of library: either "default", usually bam format (the one you gave to experiment), "pshifted" pshifted reads, "ofst", "bed", "bedo" optimized bed, or "wig"
cage	a character path to library file or a <a href="#">GRanges</a> , <a href="#">GAlignments</a> preloaded file of CAGE data. Only used if "TSS" is defined as region, to redefine 5' leaders.
format	a character, default ".png", alternative ".pdf"
acceptedLengths	an integer vector (NULL), the read lengths accepted. Default NULL, means all lengths accepted.
upstream	1 or 2 integers, default c(50, 30), how long upstream from 0 should window extend (first index is 5' end extension, second is 3' end extension). If only 1 shifting, only 1 value should be given, if two are given will use first.
downstream	1 or 2 integers, default c(29, 69), how long upstream from 0 should window extend (first index is 5' end extension, second is 3' end extension). If only 1 shifting, only 1 value should be given, if two are given will use first.
shifting	a character, default c("5prime", "3prime"), can also be either or NULL (no shifting of reads)

### Value

invisible(NULL), plots are saved

### See Also

Other heatmaps: [coverageHeatMap\(\)](#), [heatMapL\(\)](#), [heatMap\\_single\(\)](#)

**Examples**

```
df <- ORFik.template.experiment()[3,] # Only third library
#heatMapRegion(df, "TIS", outdir = "PATH/TO/SAVE/")
# Do also TSS, add cage for specific TSS
# heatMapRegion(df, c("TSS", "TIS"), cage = "path/to/cage.bed")

# Do on pshifted reads instead of original files
# outputLibs(df, type = "pshifted")
# heatMapRegion(df, "TIS")
```

---

heatMap_single	<i>Coverage heatmap of single libraries</i>
----------------	---

---

**Description**

Coverage heatmap of single libraries

**Usage**

```
heatMap_single(
  region,
  tx,
  reads,
  outdir,
  scores = "sum",
  upstream,
  downstream,
  zeroPosition = upstream,
  returnCoverage = FALSE,
  acceptedLengths = NULL,
  legendPos = "right",
  colors = "default",
  addFracPlot = TRUE,
  location = "start site",
  shifting = NULL,
  skip.last = FALSE,
  title = NULL
)
```

**Arguments**

region	#' a <a href="#">GRangesList</a> object of region, usually either leaders, cds', 3' utrs or ORFs, start region, stop regions etc. This is the region that will be mapped in heatmap
tx	default NULL, a <a href="#">GRangesList</a> of transcripts or (container region), names of tx must contain all grl names. The names of grl can also be the ORFik orf names. that is "txName_id"
reads	a <a href="#">GAlignments</a> or <a href="#">GRanges</a> object of RiboSeq, RnaSeq etc. Weights for scoring is default the 'score' column in 'reads'
outdir	a character path to save file as: not just directory, but full name.

scores	character vector, default "sum", either of zscore, transcriptNormalized, sum, mean, median, .. see ?coverageScorings for info and more alternatives.
upstream	an integer, relative region to get upstream from.
downstream	an integer, relative region to get downstream from
zeroPosition	an integer DEFAULT (upstream), what is the center point? Like leaders and cds combination, then 0 is the TIS and -1 is last base in leader. NOTE!: if windows have different widths, this will be ignored.
returnCoverage	logical, default: FALSE, return coverage, if FALSE returns plot instead.
acceptedLengths	an integer vector (NULL), the read lengths accepted. Default NULL, means all lengths accepted.
legendPos	a character, Default "right". Where should the fill legend be ? ("top", "bottom", "right", "left")
colors	character vector, default: "default", this gives you: c("white", "yellow2", "yellow3", "lightblue", "blue", "navy"), do "high" for more high contrasts, or specify your own colors.
addFracPlot	Add margin histogram plot on top of heatmap with fractions per positions
location	a character, default "start site", will make xlabel of heatmap be Position relative to "start site" or alternative given.
shifting	a character, default NULL (no shifting), can also be either of c("5prime", "3prime")
skip.last	skip top(highest) read length, default FALSE
title	a character, default NULL (no title), what is the top title of plot?

**Value**

ggplot2 grob (default), data.table (if returnCoverage is TRUE)

**See Also**

Other heatmaps: [coverageHeatMap\(\)](#), [heatMapL\(\)](#), [heatMapRegion\(\)](#)

---

import.bedo

*Load GRanges object from .bedo*

---

**Description**

.bedo is .bed ORFik, an optimized bed format for coverage reads with read lengths .bedo is a text based format with columns (6 maximum):

1. chromosome
2. start
3. end
4. strand
5. ref width (cigar # M's, match/mismatch total)
6. duplicates of that read

**Usage**

import.bedo(path)



**Arguments**

path                    a character, location on disc (full path)

**Details**

Positions are 1-based, not 0-based as .bed. export with export.bedo

**Value**

GRanges object

---

import.bedoc	<i>Load GAlignments object from .bedoc</i>
--------------	--

---

**Description**

A much faster way to store, load and use bam files.  
 .bedoc is .bed ORFik, an optimized bed format for coverage reads with cigar and replicate number.  
 .bedoc is a text based format with columns (5 maximum):

1. chromosome
2. cigar: (cigar # M's, match/mismatch total)
3. start (left most position)
4. strand (+, -, \*)
5. score: duplicates of that read

**Usage**

```
import.bedoc(path)
```

**Arguments**

path                    a character, location on disc (full path)

**Details**

Positions are 1-based, not 0-based as .bed. export with export.bedo

**Value**

GAlignments object

import.ofst

*Load GRanges / GAlignments object from .ofst***Description**

A much faster way to store, load and use bam files.

.ofst is ORFik fast serialized object, an optimized format for coverage reads with cigar and replicate number. It uses the fst format as back-end: [fst-package](#).

A .ofst ribo seq file can compress the information in a bam file from 5GB down to a few MB. This new files has super fast reading time, only a few seconds, instead of minutes. It also has random index access possibility of the file.

.ofst is represented as a data.frame format with minimum 4 columns:

1. chromosome
2. start (left most position)
3. strand (+, -, \*)
4. width (not added if cigar exists)
5. cigar (not needed if width exists): (cigar # M's, match/mismatch total)
5. score: duplicates of that read
6. size: qwidth according to reference of read

If file is from [GAlignmentPairs](#), it will contain a cigar1, cigar2 instead of cigar and start1 and start2 instead of start

**Usage**

```
import.ofst(file)
```

**Arguments**

```
file          a path to a .ofst file
```

**Details**

Other columns can be named whatever you want and added to meta columns. Positions are 1-based, not 0-based as .bed. Import with import.ofst

**Value**

a GAlignment, GAlignmentPairs or GRanges object, dependent of if cigar/cigar1 is defined in .ofst file.

**Examples**

```
## GRanges
gr <- GRanges("1:1-3:-")
# export.ofst(gr, file = "path.ofst")
# import.ofst("path.ofst")
## GAlignment
# Make input data.frame
df <- data.frame(seqnames = "1", cigar = "3M", start = 1L, strand = "+")
ga <- ORFik::getGAlignments(df)
# export.ofst(ga, file = "path.ofst")
# import.ofst("path.ofst")
```

---

```
importGtfFromTxdb      Import the GTF / GFF that made the txdb
```

---

**Description**

Import the GTF / GFF that made the txdb

**Usage**

```
importGtfFromTxdb(txdb)
```

**Arguments**

txdb                    a TxDb, path to txdb / gff or ORFik experiment object

**Value**

data.frame, the gtf/gff object imported with rtracklayer::import

---

```
initiationScore      Get initiation score for a GRangesList of ORFs
```

---

**Description**

initiationScore tries to check how much each TIS region resembles, the average of the CDS TIS regions.

**Usage**

```
initiationScore(grl, cds, tx, reads, pShifted = TRUE, weight = "score")
```

**Arguments**

grl                    a [GRangesList](#) object with ORFs  
 cds                    a [GRangesList](#) object with coding sequences  
 tx                    a [GrangesList](#) of transcripts covering grl.  
 reads                 ribo seq reads as [GAlignments](#), [GRanges](#) or [GRangesList](#) object  
 pShifted             a logical (TRUE), are riboseq reads p-shifted?  
 weight                a vector (default: 1L, if 1L it is identical to countOverlaps()), if single number (!= 1), it applies for all, if more than one must be equal size of 'reads'. else it must be the string name of a defined meta column in subject "reads", that gives number of times a read was found. [GRanges](#)("chr1", 1, "+", score = 5), would mean "score" column tells that this alignment region was found 5 times.

**Details**

Since this features uses a distance matrix for scoring, values are distributed like this: As result there is one value per ORF: 0.000: means that ORF had no reads -1.000: means that ORF is identical to average of CDS 1.000: means that orf is maximum different than average of CDS

If a score column is defined, it will use it as weights, see [getWeights](#)

**Value**

an integer vector, 1 score per ORF, with names of grl

**References**

doi: 10.1186/s12915-017-0416-0

**See Also**

Other features: [computeFeaturesCage\(\)](#), [computeFeatures\(\)](#), [countOverlapsW\(\)](#), [disengagementScore\(\)](#), [distToCds\(\)](#), [distToTSS\(\)](#), [entropy\(\)](#), [floss\(\)](#), [fpkm\\_calc\(\)](#), [fpkm\(\)](#), [fractionLength\(\)](#), [insideOutsideORF\(\)](#), [isInFrame\(\)](#), [isOverlapping\(\)](#), [kozakSequenceScore\(\)](#), [orfScore\(\)](#), [rankOrder\(\)](#), [ribosomeReleaseScore\(\)](#), [ribosomeStallingScore\(\)](#), [startRegionCoverage\(\)](#), [startRegion\(\)](#), [stopRegion\(\)](#), [subsetCoverage\(\)](#), [translationalEff\(\)](#)

**Examples**

```
# Good hitting ORF
ORF <- GRanges(seqnames = "1",
               ranges = IRanges(21, 40),
               strand = "+")
names(ORF) <- c("tx1")
grl <- GRangesList(tx1 = ORF)
# 1 width p-shifted reads
reads <- GRanges("1", IRanges(c(21, 23, 50, 50, 50, 53, 53, 56, 59),
                              width = 1), "+")

score(reads) <- 28 # original width
cds <- GRanges(seqnames = "1",
               ranges = IRanges(50, 80),
               strand = "+")
cds <- GRangesList(tx1 = cds)
tx <- GRanges(seqnames = "1",
              ranges = IRanges(1, 85),
              strand = "+")
tx <- GRangesList(tx1 = tx)

initiationScore(grl, cds, tx, reads, pShifted = TRUE)
```

---

insideOutsideORF      *Inside/Outside score (IO)*

---

**Description**

Inside/Outside score is defined as

$$(\text{reads over ORF}) / (\text{reads outside ORF and within transcript})$$

A pseudo-count of one is added to both the ORF and outside sums.



```

grl <- GRangesList(tx1_1 = ORF)

tx1 <- GRanges(seqnames = "1",
               ranges = IRanges(start = c(1, 10, 20, 30, 40, 50),
                                end = c(5, 15, 25, 35, 45, 200)),
               strand = "+")
tx <- GRangesList(tx1 = tx1)
RFP <- GRanges(seqnames = "1",
               ranges = IRanges(start = c(1, 4, 30, 60, 80, 90),
                                end = c(30, 33, 63, 90, 110, 120)),
               strand = "+")

insideOutsideORF(grl, RFP, tx)

```

---

install.fastp

*Download and prepare fastp trimmer*


---

### Description

On Linux, will not run "make", only use precompiled fastp file.  
 On Mac OS it will use precompiled binaries.  
 Does not work yet for Windows!

### Usage

```
install.fastp(folder = "~/bin")
```

### Arguments

folder            path to folder for download, file will be named "fastp", this should be most recent version. On mac it will search for a folder called fastp-master inside folder given. Since there is no precompiled version of fastp for Mac OS.

### Value

path to runnable fastp

### References

<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6129281/>

### See Also

Other STAR: `STAR.align.folder()`, `STAR.align.single()`, `STAR.allsteps.multiQC()`, `STAR.index()`, `STAR.install()`, `STAR.multiQC()`, `STAR.remove.crashed.genome()`, `getGenomeAndAnnotation()`

### Examples

```
#install.fastp()
```

---

install.sratoolkit      *Download sra toolkit*

---

**Description**

Download sra toolkit

**Usage**

```
install.sratoolkit(folder = "~/bin", version = "2.10.8")
```

**Arguments**

folder	default folder, "~/bin"
version	a string, default "2.10.8"

**Value**

path to fastq-dump in sratoolkit

**References**

<https://ncbi.github.io/sra-tools/fastq-dump.html>

**Examples**

```
# install.sratoolkit()
## Custom version
# install.sratoolkit(version = "2.10.7")
```

---

is.grl      *Helper function to check for GRangesList*

---

**Description**

Helper function to check for GRangesList

**Usage**

```
is.grl(class)
```

**Arguments**

class	the class you want to check if is GRL, either a character from class or the object itself.
-------	--

**Value**

a boolean

**See Also**

Other validity: [checkRFP\(\)](#), [checkRNA\(\)](#), [is.ORF\(\)](#), [is.gr\\_or\\_grl\(\)](#), [is.range\(\)](#), [validGRL\(\)](#), [validSeqlevels\(\)](#)

---

is.gr\_or\_grl

*Helper function to check for GRangesList or GRanges class*

---

**Description**

Helper function to check for GRangesList or GRanges class

**Usage**

```
is.gr_or_grl(class)
```

**Arguments**

class            the class you want to check if is GRL or GR, either a character from class or the object itself.

**Value**

a boolean

**See Also**

Other validity: [checkRFP\(\)](#), [checkRNA\(\)](#), [is.ORF\(\)](#), [is.grl\(\)](#), [is.range\(\)](#), [validGRL\(\)](#), [validSeqlevels\(\)](#)

---

is.ORF

*Check if all requirements for an ORFik ORF is accepted.*

---

**Description**

Check if all requirements for an ORFik ORF is accepted.

**Usage**

```
is.ORF(grl)
```

**Arguments**

grl              a GRangesList or GRanges to check

**Value**

a logical (TRUE/FALSE)

**See Also**

Other validity: [checkRFP\(\)](#), [checkRNA\(\)](#), [is.gr\\_or\\_grl\(\)](#), [is.grl\(\)](#), [is.range\(\)](#), [validGRL\(\)](#), [validSeqlevels\(\)](#)



---

is.range	<i>Helper function to check for ranged object</i>
----------	---

---

**Description**

Helper function to check for ranged object

**Usage**

```
is.range(class)
```

**Arguments**

class	the class you want to check if is GRL or GR, either a character from class or the object itself.
-------	--

**Value**

a boolean

**See Also**

Other validity: [checkRFP\(\)](#), [checkRNA\(\)](#), [is.ORF\(\)](#), [is.gr\\_or\\_grl\(\)](#), [is.grl\(\)](#), [validGRL\(\)](#), [validSeqlevels\(\)](#)

---

isInFrame	<i>Find frame for each orf relative to cds</i>
-----------	--

---

**Description**

Input of this function, is the output of the function [[distToCds\(\)](#)], or any other relative ORF frame.

**Usage**

```
isInFrame(dists)
```

**Arguments**

dists	a vector of integer distances between ORF and cds. 0 distance means equal frame
-------	---

**Details**

possible outputs: 0: orf is in frame with cds 1: 1 shifted from cds 2: 2 shifted from cds

**Value**

a logical vector

**References**

doi: 10.1074/jbc.R116.733899

**See Also**

Other features: [computeFeaturesCage\(\)](#), [computeFeatures\(\)](#), [countOverlapsW\(\)](#), [disengagementScore\(\)](#), [distToCds\(\)](#), [distToTSS\(\)](#), [entropy\(\)](#), [floss\(\)](#), [fpkm\\_calc\(\)](#), [fpkm\(\)](#), [fractionLength\(\)](#), [initiationScore\(\)](#), [insideOutsideORF\(\)](#), [isOverlapping\(\)](#), [kozakSequenceScore\(\)](#), [orfScore\(\)](#), [rankOrder\(\)](#), [ribosomeReleaseScore\(\)](#), [ribosomeStallingScore\(\)](#), [startRegionCoverage\(\)](#), [startRegion\(\)](#), [stopRegion\(\)](#), [subsetCoverage\(\)](#), [translationalEff\(\)](#)

**Examples**

```
# simple example
isInFrame(c(3,6,8,11,15))

# GRangesList example
gr1 <- GRangesList(tx1_1 = GRanges("1", IRanges(1,10), "+"))
fiveUTRs <- GRangesList(tx1 = GRanges("1", IRanges(1,20), "+"))
dist <- distToCds(gr1, fiveUTRs)
isInFrame <- isInFrame(dist)
```

---

isOverlapping

*Find frame for each orf relative to cds*

---

**Description**

Input of this function, is the output of the function [distToCds()]

**Usage**

```
isOverlapping(dists)
```

**Arguments**

dists                    a vector of distances between ORF and cds

**Value**

a logical vector

**References**

doi: 10.1074/jbc.R116.733899

**See Also**

Other features: [computeFeaturesCage\(\)](#), [computeFeatures\(\)](#), [countOverlapsW\(\)](#), [disengagementScore\(\)](#), [distToCds\(\)](#), [distToTSS\(\)](#), [entropy\(\)](#), [floss\(\)](#), [fpkm\\_calc\(\)](#), [fpkm\(\)](#), [fractionLength\(\)](#), [initiationScore\(\)](#), [insideOutsideORF\(\)](#), [isInFrame\(\)](#), [kozakSequenceScore\(\)](#), [orfScore\(\)](#), [rankOrder\(\)](#), [ribosomeReleaseScore\(\)](#), [ribosomeStallingScore\(\)](#), [startRegionCoverage\(\)](#), [startRegion\(\)](#), [stopRegion\(\)](#), [subsetCoverage\(\)](#), [translationalEff\(\)](#)

**Examples**

```
# simple example
isOverlapping(c(-3,-6,8,11,15))

# GRangesList example
grl <- GRangesList(tx1_1 = GRanges("1", IRanges(1,10), "+"))
fiveUTRs <- GRangesList(tx1 = GRanges("1", IRanges(1,20), "+"))
dist <- distToCds(grl, fiveUTRs)
isOverlapping <- isOverlapping(dist)
```

---

isPeriodic	<i>Find if there is a periodicity of 3 in the vector</i>
------------	--

---

**Description**

It uses Fourier transform + periodogram for finding periodic vectors on the transcript normalized counts over all CDS regions from 0 to 149, where TIS is 0.  
Checks if there is a periodicity and if the periodicity is 3, more precisely between 2.9 and 3.1.

**Usage**

```
isPeriodic(x)
```

**Arguments**

x (numeric) Vector of values to detect periodicity of 3 like in RiboSeq data.

**Details**

Transcript normalized means per CDS TIS region, count reads per position, divide that number per position by the total of that transcript, then sum up these numbers per position for all transcripts.

**Value**

a logical, if it is periodic.

---

kozakHeatmap	<i>Make sequence region heatmap relative to scoring</i>
--------------	---

---

**Description**

Given sequences, DNA or RNA. And some score, ribo-seq fpkm, TE etc. Create a heatmap divided per letter in seqs, by how strong the score is.

**Usage**

```
kozakHeatmap(
  seqs,
  rate,
  start = 1,
  stop = max(nchar(seqs)),
  center = ceiling((stop - start + 1)/2),
  min.observations = ">q1",
  skip.startCodon = FALSE,
  xlab = "TIS",
  type = "ribo-seq"
)
```

**Arguments**

seqs	the sequences (character vector, DNAStringSet)
rate	a scoring vector (equal size to seqs)
start	position in seqs to start at (first is 1), default 1.
stop	position in seqs to stop at (first is 1), default max(nchar(seqs)), that is the longest sequence length
center	position in seqs to center at (first is 1), center will be +1 in heatmap
min.observations	How many observations per position per letter to accept? numeric or quantile, default (">q1", bigger than quartile 1 (25 percentile)). You can do (10), to get all with more than 10 observations.
skip.startCodon	startCodon is defined as after centering (position 1, 2 and 3). Should they be skipped? default (FALSE). Not relevant if you are not doing Translation initiation sites (TIS).
xlab	Region you are checking, default (TIS)
type	What type is the rate scoring? default (ribo-seq)

**Details**

It will create blocks around the highest rate per position

**Value**

a ggplot of the heatmap

**Examples**

```
## Not run:
if (requireNamespace("BSgenome.Hsapiens.UCSC.hg19")) {
  txdbFile <- system.file("extdata", "hg19_knownGene_sample.sqlite",
                          package = "GenomicFeatures")
  #Extract sequences of Coding sequences.
  cds <- loadRegion(txdbFile, "cds")
  tx <- loadRegion(txdbFile, "mrna")
}
```

```

# Get region to check
kozakRegions <- startRegionString(cds, tx, BSgenome.Hsapiens.UCSC.hg19::Hsapiens
                                , upstream = 4, 5)
# Some toy ribo-seq fpkm scores on cds
set.seed(3)
fpkm <- sample(1:115, length(cds), replace = TRUE)
kozakHeatmap(kozakRegions, fpkm, 1, 9, skip.startCodon = F)
}

## End(Not run)

```

---

kozakSequenceScore      *Make a score for each ORFs start region by proximity to Kozak*

---

### Description

The closer the sequence is to the Kozak sequence the higher the score, based on the experimental pwm from article referenced. Minimum score is 0 (worst correlation), max is 1 (the best base per column was chosen).

### Usage

```
kozakSequenceScore(gr1, tx, faFile, species = "human", include.N = FALSE)
```

### Arguments

gr1	a <a href="#">GRangesList</a> grouped by ORF
tx	a <a href="#">GRangesList</a> , the reference area for ORFs, each ORF must have a corresponding tx.
faFile	<a href="#">FaFile</a> , <a href="#">BSgenome</a> , fasta/index file path or an <a href="#">ORFik experiment</a> . This file is usually used to find the transcript sequences from some <a href="#">GRangesList</a> .
species	("human"), which species to use, currently supports human ( <i>Homo sapiens</i> ), zebrafish ( <i>Danio rerio</i> ) and mouse ( <i>Mus musculus</i> ). Both scientific or common name for these species will work. You can also specify a pfm for your own species. Syntax of pfm is a rectangular integer matrix, where all columns must sum to the same value, normally 100. See example for more information. Rows are in order: c("A", "C", "G", "T")
include.N	logical (F), if TRUE, allow N bases to be counted as hits, score will be average of the other bases. If True, N bases will be added to pfm, automatically, so dont include them if you make your own pfm.

### Details

Ranges that does not have minimum 15 length (the kozak requirement as a sliding window of size 15 around gr1 start), will be set to score 0. Since they should not have the possibility to make an efficient ribosome binding.

### Value

a numeric vector with values between 0 and 1  
 an integer vector, one score per orf

## References

doi: <https://doi.org/10.1371/journal.pone.0108475>

## See Also

Other features: [computeFeaturesCage\(\)](#), [computeFeatures\(\)](#), [countOverlapsW\(\)](#), [disengagementScore\(\)](#), [distToCds\(\)](#), [distToTSS\(\)](#), [entropy\(\)](#), [floss\(\)](#), [fpkm\\_calc\(\)](#), [fpkm\(\)](#), [fractionLength\(\)](#), [initiationScore\(\)](#), [insideOutsideORF\(\)](#), [isInFrame\(\)](#), [isOverlapping\(\)](#), [orfScore\(\)](#), [rankOrder\(\)](#), [ribosomeReleaseScore\(\)](#), [ribosomeStallingScore\(\)](#), [startRegionCoverage\(\)](#), [startRegion\(\)](#), [stopRegion\(\)](#), [subsetCoverage\(\)](#), [translationalEff\(\)](#)

## Examples

```
# Usually the ORFs are found in orfik, which makes names for you etc.
# Here we make an example from scratch
seqName <- "Chromosome"
ORF1 <- GRanges(seqnames = seqName,
                ranges = IRanges(c(1007, 1096), width = 60),
                strand = c("+", "+"))
ORF2 <- GRanges(seqnames = seqName,
                ranges = IRanges(c(400, 100), width = 30),
                strand = c("-", "-"))
ORFs <- GRangesList(tx1 = ORF1, tx2 = ORF2)
ORFs <- makeORFNames(ORFs) # need ORF names
tx <- extendLeaders(ORFs, 100)
# get faFile for sequences
faFile <- FaFile(system.file("extdata", "genome.fasta", package = "ORFik"))
kozakSequenceScore(ORFs, tx, faFile)
# For more details see vignettes.
```

---

kozak\_IR\_ranking

*Rank kozak initiation sequences*

---

## Description

Defined as region (-4, -1) relative to TIS

## Usage

```
kozak_IR_ranking(cds_k, mrna, dt.ir, faFile, group.min = 10, species = "human")
```

## Arguments

cds_k	cds ranges (GRangesList)
mrna	mrna ranges (GRangesList)
dt.ir	data.table with a column called IR, initiation rate
faFile	<a href="#">FaFile</a> , BSgenome, fasta/index file path or an <a href="#">ORFik experiment</a> . This file is usually used to find the transcript sequences from some GRangesList.
group.min	numeric, default 10. Minimum transcripts per initiation group to be included

species ("human"), which species to use, currently supports human (*Homo sapiens*), zebrafish (*Danio rerio*) and mouse (*Mus musculus*). Both scientific or common name for these species will work. You can also specify a pfm for your own species. Syntax of pfm is an rectangular integer matrix, where all columns must sum to the same value, normally 100. See example for more information. Rows are in order: c("A", "C", "G", "T")

### Value

a ggplot grid object

---

lastExonEndPerGroup    *Get last end per granges group*

---

### Description

Get last end per granges group

### Usage

```
lastExonEndPerGroup(gr1, keep.names = TRUE)
```

### Arguments

gr1                    a [GRangesList](#)

keep.names            a boolean, keep names or not, default: (TRUE)

### Value

a Rle(keep.names = T), or integer vector(F)

### Examples

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
                  ranges = IRanges(c(7, 14), width = 3),
                  strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
                  ranges = IRanges(c(4, 1), c(9, 3)),
                  strand = c("-", "-"))
gr1 <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
lastExonEndPerGroup(gr1)
```

---

lastExonPerGroup      *Get last exon per GRangesList group*

---

**Description**

grl must be sorted, call ORFik:::sortPerGroup if needed

**Usage**

```
lastExonPerGroup(grl)
```

**Arguments**

grl                    a [GRangesList](#)

**Value**

a GRangesList of the last exon per group

**Examples**

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
  ranges = IRanges(c(7, 14), width = 3),
  strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
  ranges = IRanges(c(4, 1), c(9, 3)),
  strand = c("-", "-"))
grl <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
lastExonPerGroup(grl)
```

---

lastExonStartPerGroup      *Get last start per granges group*

---

**Description**

Get last start per granges group

**Usage**

```
lastExonStartPerGroup(grl, keep.names = TRUE)
```

**Arguments**

grl                    a [GRangesList](#)  
 keep.names            a boolean, keep names or not, default: (TRUE)

**Value**

a Rle(keep.names = T), or integer vector(F)



**Examples**

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
                  ranges = IRanges(c(7, 14), width = 3),
                  strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
                   ranges = IRanges(c(4, 1), c(9, 3)),
                   strand = c("-", "-"))
grl <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
lastExonStartPerGroup(grl)
```

---

libNames	<i>Get library name variants</i>
----------	----------------------------------

---

**Description**

Used to standardize nomenclature for experiments.

Example: RFP is main naming, but a variant is ribo-seq ribo-seq will then be renamed to RFP

**Usage**

```
libNames()
```

**Value**

a data.table with 2 columns, the main name, and all name variants of the main name in second column as a list.

**See Also**

Other experiment\_naming: [cellLineNames\(\)](#), [conditionNames\(\)](#), [mainNames\(\)](#), [repNames\(\)](#), [stageNames\(\)](#), [tissueNames\(\)](#)

---

libraryTypes	<i>Which type of library type in <a href="#">experiment?</a></i>
--------------	--

---

**Description**

Which type of library type in [experiment?](#)

**Usage**

```
libraryTypes(df)
```

**Arguments**

df                    an ORFik [experiment](#)

**Value**

library types (character vector)

**See Also**

Other ORFik\_experiment: `ORFik.template.experiment()`, `bamVarName()`, `create.experiment()`, `experiment-class`, `filepath()`, `organism.df()`, `outputLibs()`, `read.experiment()`, `save.experiment()`, `validateExperiments()`

---

list.experiments	<i>List current experiment available</i>
------------------	--

---

**Description**

Will only search .csv extension, also exclude any experiment with the word template.

**Usage**

```
list.experiments(
  dir = "~/Bio_data/ORFik_experiments/",
  pattern = "*",
  libtypeExclusive = NULL,
  BPPARAM = bpparam()
)
```

**Arguments**

dir	directory for ORFik experiments: default: "~/Bio_data/ORFik_experiments/"
pattern	allowed patterns in experiment file name: default ("*", all experiments)
libtypeExclusive	search for experiments with exclusively this libtype, default (NULL, all)
BPPARAM	how many cores/threads to use? default: bpparam()

**Value**

a data.table, 1 row per experiment with columns experiment (name), libtypes

**Examples**

```
## Make your experiments
df <- ORFik.template.experiment(TRUE)
df2 <- df[1:6,] # Only first 2 libs
## Save them
# save.experiment(df, "~/Bio_data/ORFik_experiments/exp1.csv")
# save.experiment(df2, "~/Bio_data/ORFik_experiments/exp1_subset.csv")
## List all experiment you have:
## Path above is default path, so no dir argument needed
#list.experiments()
#list.experiments(pattern = "subset")
## For non default directory experiments
#list.experiments(dir = "MY/CUSTOM/PATH")
```

---

loadRegion	<i>Load transcript region</i>
------------	-------------------------------

---

**Description**

Usefull to simplify loading of standard regions, like cds' and leaders.

**Usage**

```
loadRegion(txdb, part = "tx", names.keep = NULL, by = "tx")
```

**Arguments**

txdb	a TxDb file or a path to one of: (.gtf, .gff, .gff2, .db or .sqlite), if it is a GRangesList, it will return it self.
part	a character, one of: tx, leader, cds, trailer, intron, mrna NOTE: difference between tx and mrna is that tx are all transcripts, while mrna are all transcripts with a cds
names.keep	a character vector of subset of names to keep. Example: loadRegions(txdb, names = ENST1000005), will return only that transcript. Remember if you set by to "gene", then this list must be with gene names.
by	a character, default "tx" Either "tx" or "gene". What names to output region by, the transcript name "tx" or gene names "gene"

**Details**

Load as GRangesList if input is not already GRangesList.

**Value**

a GrangesList of region

**Examples**

```
gtf <- system.file("extdata", "annotations.gtf", package = "ORFik")
loadRegion(gtf, "cds")
loadRegion(gtf, "intron")
```

---

loadRegions	<i>Get all regions of transcripts specified to environment</i>
-------------	--

---

**Description**

By default loads all parts to .GlobalEnv (global environemnt) Useful to not spend time on finding the functions to load regions.

**Usage**

```
loadRegions(
  txdb,
  parts = c("mrna", "leaders", "cds", "trailers"),
  extension = "",
  names.keep = NULL,
  by = "tx",
  envir = .GlobalEnv
)
```

**Arguments**

txdb	a TxDb file, a path to one of: (.gtf, .gff, .gff2, .db or .sqlite) or an ORFik experiment
parts	the transcript parts you want, default: c("mrna", "leaders", "cds", "trailers"). See ?loadRegion for more info on this argument.
extension	What to add on the name after leader, like: B -> leadersB
names.keep	a character vector of subset of names to keep. Example: loadRegions(txdb, names = ENST1000005), will return only that transcript. Remember if you set by to "gene", then this list must be with gene names.
by	a character, default "tx" Either "tx" or "gene". What names to output region by, the transcript name "tx" or gene names "gene"
envir	Which environment to save to, default: .GlobalEnv

**Value**

invisible(NULL) (regions saved in envir)

**Examples**

```
# Load all mrna regions to Global environment
gtf <- system.file("extdata", "annotations.gtf", package = "ORFik")
loadRegions(gtf, parts = c("mrna", "leaders", "cds", "trailers"))
```

---

loadTranscriptType      *Load transcripts of given biotype*

---

**Description**

Like rRNA, snoRNA etc. NOTE: Only works on gtf/gff, not .db object for now. Also note that these annotations are not perfect, some rRNA annotations only contain 5S rRNA etc. If your gtf does not contain everything you need, use a resource like repeatmasker and download a gtf: <https://genome.ucsc.edu/cgi-bin/hgTables>

**Usage**

```
loadTranscriptType(object, part = "rRNA", tx = NULL)
```

**Arguments**

object	a TxDb, ORFik experiment or path to gtf/gff,
part	a character, default rRNA. Can also be: snoRNA, tRNA etc. As long as that biotype is defined in the gtf.
tx	a GRangesList of transcripts (Optional, default NULL, all transcript of that type), else it must be names a list to subset on.

**Value**

a GRangesList of transcript of that type

**References**

doi: 10.1002/0471250953.bi0410s25

---

loadTxdb	<i>General loader for txdb</i>
----------	--------------------------------

---

**Description**

Useful to allow fast TxDb loader like .db

**Usage**

```
loadTxdb(txdb, chrStyle = NULL)
```

**Arguments**

txdb	a TxDb file, a path to one of: (.gtf, .gff, .gff2, .gff2, .db or .sqlite) or an ORFik experiment
chrStyle	a GRanges object, TxDb, FaFile, or a <a href="#">seqlevelsStyle</a> (Default: NULL) to get seqlevelsStyle from. Is chromosome 1 called chr1 or 1, is mitochondrial chromosome called MT or chrM etc. Will use 1st seqlevel-style if more are present. Like: c("NCBI", "UCSC") -> pick "NCBI"

**Value**

a TxDb object

**Examples**

```
library(GenomicFeatures)
# Get the gtf txdb file
txdbFile <- system.file("extdata", "hg19_knownGene_sample.sqlite",
                        package = "GenomicFeatures")
txdb <- loadDb(txdbFile)
```

---

longestORFs                      *Get longest ORF per stop site*

---

### Description

Rule: if seqname, strand and stop site is equal, take longest one. Else keep. If IRangesList or IRanges, seqnames are groups, if GRanges or GRangesList seqnames are the seqlevels (e.g. chromosomes/transcripts)

### Usage

```
longestORFs(gr1)
```

### Arguments

gr1                      a [GRangesList](#)/[IRangesList](#), [GRanges](#)/[IRanges](#) of ORFs

### Value

a [GRangesList](#)/[IRangesList](#), [GRanges](#)/[IRanges](#) (same as input)

### See Also

Other ORFHelpers: [defineTrailer\(\)](#), [mapToGRanges\(\)](#), [orfID\(\)](#), [startCodons\(\)](#), [startSites\(\)](#), [stopCodons\(\)](#), [stopSites\(\)](#), [txNames\(\)](#), [uniqueGroups\(\)](#), [uniqueOrder\(\)](#)

### Examples

```
ORF1 = GRanges("1", IRanges(10,21), "+")
ORF2 = GRanges("1", IRanges(1,21), "+") # <- longest
gr1 <- GRangesList(ORF1 = ORF1, ORF2 = ORF2)
longestORFs(gr1) # get only longest
```

---

mainNames                      *Get main name from variant name*

---

### Description

Used to standardize nomenclature for experiments.

Example: RFP is main naming, but a variant is ribo-seq ribo-seq will then be renamed to RFP

### Usage

```
mainNames(names, dt)
```

### Arguments

names                      a character vector of names that must exist in dt\$allNames  
 dt                          a data.table with 2 columns (mainName, allNames)

**Value**

a data.table with 2 columns, the main name, and all name variants of the main name in second column as a list.

**See Also**

Other experiment\_naming: [cellLineNames\(\)](#), [conditionNames\(\)](#), [libNames\(\)](#), [repNames\(\)](#), [stageNames\(\)](#), [tissueNames\(\)](#)

---

makeExonRanks	<i>Make grouping by exons ranks</i>
---------------	-------------------------------------

---

**Description**

There are two ways to make vector of exon ranking: 1. Iterate per exon in ORF, byTranscript = FALSE 2. Iterate per ORF in transcript, byTranscript = TRUE.

**Usage**

```
makeExonRanks(gr1, byTranscript = FALSE)
```

**Arguments**

gr1	a <a href="#">GRangesList</a>
byTranscript	logical (default: FALSE), groups orfs by transcript name or ORF name, if ORfs are by transcript, check duplicates.

**Details**

Either by transcript or by original groupings. Must be ordered, so that same transcripts are ordered together.

**Value**

an integer vector of indices for exon ranks

---

makeORFNames	<i>Make ORF names per orf</i>
--------------	-------------------------------

---

**Description**

gr1 must be grouped by transcript If a list of orfs are grouped by transcripts, but does not have ORF names, then create them and return the new GRangesList

**Usage**

```
makeORFNames(gr1, groupByTx = TRUE)
```

**Arguments**

`gr1` a [GRangesList](#)

`groupByTx` logical (T), should output GRangesList be grouped by transcripts (T) or by ORFs (F)?

**Value**

(GRangesList) with ORF names, grouped by transcripts, sorted.

**Examples**

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
  ranges = IRanges(c(7, 14), width = 3),
  strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
  ranges = IRanges(c(4, 1), c(9, 3)),
  strand = c("-", "-"))
gr1 <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
makeORFNames(gr1)
```

---

makeSummarizedExperimentFromBam

*Make a count matrix from a library or experiment*

---

**Description**

Make a summarizedExperiment / matrix object from bam files

**Usage**

```
makeSummarizedExperimentFromBam(
  df,
  saveName = NULL,
  longestPerGene = TRUE,
  geneOrTxNames = "tx",
  region = "mrna",
  type = "count",
  weight = "score"
)
```

**Arguments**

`df` an ORFik [experiment](#)

`saveName` a character (default NULL), if set save experiment to path given. Always saved as .rds., it is optional to add .rds, it will be added for you if not present. Also used to load existing file with that name.

`longestPerGene` a logical (default TRUE), if FALSE all transcript isoforms per gene.

`geneOrTxNames` a character vector (default "tx"), should row names keep transcript names ("tx") or change to gene names ("gene")



region	a character vector (default: "mrna"), make raw count matrices of whole mrnas or one of (leaders, cds, trailers). Can also be a <a href="#">GRangesList</a> , then it uses this region directly.
type	default: "count" (raw counts matrix), alternative is "fpkm", "log2fpkm" or "log10fpkm"
weight	numeric or character, a column to score overlaps by. Default "score", will check for a metacolumn called "score" in libraries. If not found, will not use weights.

### Details

If txdb or gtf path is added, it is a rangedSummerizedExperiment NOTE: If the file called saveName exists, it will then load file, not remake it!

### Value

a [SummarizedExperiment](#) object or data.table if "type" is not "count, with rownames as transcript / gene names.

### Examples

```
##Make experiment
df <- ORFik.template.experiment()
# makeSummarizedExperimentFromBam(df)
# Only cds (coding sequences):
# makeSummarizedExperimentFromBam(df, region = "cds")
# FPKM instead of raw counts on whole mrna regions
# makeSummarizedExperimentFromBam(df, type = "fpkm")
```

---

mapToGRanges	<i>Map orfs to genomic coordinates</i>
--------------	--

---

### Description

Creates GRangesList from the results of ORFs\_as\_List and the GRangesList used to find the ORFs

### Usage

```
mapToGRanges(grl, result, groupByTx = TRUE)
```

### Arguments

grl	A <a href="#">GRangesList</a> of the original sequences that gave the orfs in Genomic coordinates.
result	IRangesList A list of the results of finding uorfs list syntax is: Per list group in IRangesList is per grl index. In transcript coordinates. The names are grl index as character.
groupByTx	logical (T), should output GRangesList be grouped by transcripts (T) or by ORFs (F)?

### Details

There is no check on invalid matches, so be carefull if you use this function directly.

**Value**

A [GRangesList](#) of ORFs.

**See Also**

Other ORFHelpers: [defineTrailer\(\)](#), [longestORFs\(\)](#), [orfID\(\)](#), [startCodons\(\)](#), [startSites\(\)](#), [stopCodons\(\)](#), [stopSites\(\)](#), [txNames\(\)](#), [uniqueGroups\(\)](#), [uniqueOrder\(\)](#)

---

matchColors	<i>Match coloring of coverage plot</i>
-------------	--

---

**Description**

Check that colors match with the number of fractions.

**Usage**

```
matchColors(coverage, colors)
```

**Arguments**

coverage	a data.table with coverage
colors	a character vector of colors

**Value**

number of genes in coverage

---

matchNaming	<i>Match naming of GRangesList</i>
-------------	------------------------------------

---

**Description**

Given a [GRangesList](#) and a reference, make the naming convention and the number of metacolumns equal to reference

**Usage**

```
matchNaming(gr, reference)
```

**Arguments**

gr	a <a href="#">GRangesList</a> or GRanges object
reference	a <a href="#">GRangesList</a> of a reference

**Value**

a [GRangesList](#)

---

matchSeqStyle	<i>A wrapper for seqlevelsStyle</i>
---------------	-------------------------------------

---

**Description**

To make sure chromosome naming is correct (chr1 vs 1 vs I etc)

**Usage**

```
matchSeqStyle(range, chrStyle = NULL)
```

**Arguments**

range	a ranged object, (GRanges, GAlignment etc)
chrStyle	a GRanges object, TxDb, FaFile, or a <a href="#">seqlevelsStyle</a> (Default: NULL) to get seqlevelsStyle from. Is chromosome 1 called chr1 or 1, is mitochondrial chromosome called MT or chrM etc. Will use 1st seqlevel-style if more are present. Like: c("NCBI", "UCSC") -> pick "NCBI"

**Value**

a GAlignment/GRanges object depending on input.

---

mergeFastq	<i>Merge groups of Fastq /Fasta files</i>
------------	---

---

**Description**

Will use multithreading to speed up process. Only works for Unix OS (Linux and Mac)

**Usage**

```
mergeFastq(in_files, out_files, BPPARAM = bpparam())
```

**Arguments**

in_files	character specify the full path to the individual fastq.gz files. Seperated by space per file in group: For 2 output files from 4 input files: in_files <- c("file1.fastq file2.fastq", "file3.fastq file4.fastq")
out_files	character specify the path to the FASTQ directory For 2 output files: out_files <- c("/merged/file1&2.fastq", "/merged/file3&4.fastq")
BPPARAM	how many cores/threads to use? default: bpparam(). To see number of threads used, do bpparam() \$workers

**Value**

invisible(NULL).

## Examples

```

fastq.folder <- tempdir() # <- Your fastq files
infile <- dir(fastq.folder, "*.fastq", full.names = TRUE)
## Not run:
# Separate files into groups (here it is 4 output files from 12 input files)
in_files <- c(paste0(grep(infile, pattern = paste0("ribopool-",
  seq(11, 14), collapse = "|"), value = TRUE), collapse = " "),
  paste0(grep(infile, pattern = paste0("ribopool-",
  seq(18, 19), collapse = "|"), value = TRUE), collapse = " "),
  paste0(grep(infile, pattern = paste0("C11-",
  seq(11, 14), collapse = "|"), value = TRUE), collapse = " "),
  paste0(grep(infile, pattern = paste0("C11-",
  seq(18, 19), collapse = "|"), value = TRUE), collapse = " "))

out_files <- paste0(c("SSU_ribopool", "LSU_ribopool", "SSU_WT", "LSU_WT"), ".fastq.gz")
merged.fastq.folder <- file.path(fastq.folder, "merged/")
out_files <- file.path(merged.fastq.folder, out_files)

mergeFastq(in_files, out_files)

## End(Not run)

```

---

metaWindow

*Calculate meta-coverage of reads around input GRanges/List object.*

---

## Description

Sums up coverage over set of GRanges objects as a meta representation.

## Usage

```

metaWindow(
  x,
  windows,
  scoring = "sum",
  withFrames = FALSE,
  zeroPosition = NULL,
  scaleTo = 100,
  fraction = NULL,
  feature = NULL,
  forceUniqueEven = !is.null(scoring),
  weight = "score"
)

```

## Arguments

x	GRanges/GAlignment object of your reads. Remember to resize them beforehand to width of 1 to focus on 5' ends of footprints etc, if that is wanted.
windows	GRangesList or GRanges of your ranges
scoring	a character, default: "sum", one of (zscore, transcriptNormalized, mean, median, sum, sumLength, NULL), see ?coverageScorings for info and more alternatives.

withFrames	a logical (TRUE), return positions with the 3 frames, relative to zeroPosition. zeroPosition is frame 0.
zeroPosition	an integer DEFAULT (NULL), the point if all windows are equal size, that should be set to position 0. Like leaders and cds combination, then 0 is the TIS and -1 is last base in leader. NOTE!: if windows have different widths, this will be ignored.
scaleTo	an integer (100), if windows have different size, a meta window can not directly be created, since a meta window must have equal size for all windows. Rescale (bin) all windows to scaleTo. i.e c(1,2,3) -> size 2 -> coverage of position c(1, mean(2,3)) etc.
fraction	a character/integer (NULL), the fraction i.e (27) for read length 27, or ("LSU") for large sub-unit TCP-seq.
feature	a character string, info on region. Usually either gene name, transcript part like cds, leader, or CpG motifs etc.
forceUniqueEven,	a logical (TRUE), if TRUE; require that all windows are of same width and even. To avoid bugs. FALSE if score is NULL.
weight	(default: 'score'), if defined a character name of valid meta column in subject. GRanges("chr1", 1, "+", score = 5), would mean score column tells that this alignment region was found 5 times. ORFik .bedo files, contains a score column like this. As do CAGER CAGE files and many other package formats. You can also assign a score column manually.

## Value

A data.table with scored counts (score) of reads mapped to positions (position) specified in windows along with frame (frame).

## See Also

Other coverage: [coverageScorings\(\)](#), [regionPerReadLength\(\)](#), [scaledWindowPositions\(\)](#), [windowPerReadLength\(\)](#)

## Examples

```
library(GenomicRanges)
windows <- GRangesList(GRanges("chr1", IRanges(c(50, 100), c(80, 200)),
                                "-"))
x <- GenomicRanges::GRanges(
  seqnames = "chr1",
  ranges = IRanges::IRanges(c(100, 180), c(200, 300)),
  strand = "-")
metaWindow(x, windows, withFrames = FALSE)
```

---

nrow, experiment-method

*Internal nrow function for ORFik experiment Number of runs in experiment*

---

### Description

Internal nrow function for ORFik experiment Number of runs in experiment

### Usage

```
## S4 method for signature 'experiment'
nrow(x)
```

### Arguments

x                    an ORFik [experiment](#)

### Value

number of rows in experiment (integer)

---

numCodons

*Get number of codons*

---

### Description

Length of object / 3. Choose either only whole codons, or with stubs. For orfs stubs are not relevant, since there are no correctly defined ORFs that are 17 bases long etc.

### Usage

```
numCodons(gr1, as.integer = TRUE, keep.names = FALSE)
```

### Arguments

gr1                    a [GRangesList](#) object  
as.integer            a logical (TRUE), remove stub codons  
keep.names            a logical (FALSE)

### Value

an integer vector

---

numExonsPerGroup	<i>Get list of the number of exons per group</i>
------------------	--

---

**Description**

Can also be used generally to get number of GRanges object per GRangesList group

**Usage**

```
numExonsPerGroup(gr1, keep.names = TRUE)
```

**Arguments**

gr1	a GRangesList
keep.names	a logical, keep names or not, default: (TRUE)

**Value**

an integer vector of counts

**Examples**

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
  ranges = IRanges(c(7, 14), width = 3),
  strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
  ranges = IRanges(c(4, 1), c(9, 3)),
  strand = c("-", "-"))
gr1 <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
numExonsPerGroup(gr1)
```

---

optimizeReads	<i>Find optimized subset of valid reads</i>
---------------	---

---

**Description**

Keep only the ones that overlap within the gr1 ranges. Also sort them in the end

**Usage**

```
optimizeReads(gr1, reads)
```

**Arguments**

gr1	a <a href="#">GRangesList</a> or GRanges object
reads	a GRanges, GAlignment or GAlignmentPairs object

**Value**

the reads as GRanges, GAlignment or GAlignmentPairs

**See Also**

Other utils: [bedToGR\(\)](#), [convertToOneBasedRanges\(\)](#), [export.bed12\(\)](#), [export.wiggle\(\)](#), [fimport\(\)](#), [findFa\(\)](#), [fread.bed\(\)](#), [readBam\(\)](#), [readWig\(\)](#)

---

orfID	<i>Get id's for each orf</i>
-------	------------------------------

---

**Description**

These id's can be unqued by isoform etc, this is not supported by GenomicRanges.

**Usage**

```
orfID(grl, with.tx = FALSE)
```

**Arguments**

grl	a <a href="#">GRangesList</a>
with.tx	a boolean, include transcript names, if you want unique orfs, so that they dont have multiple versions on different isoforms, set it to FALSE.

**Value**

a character vector of ids, 1 per orf

**See Also**

Other ORFHelpers: [defineTrailer\(\)](#), [longestORFs\(\)](#), [mapToGRanges\(\)](#), [startCodons\(\)](#), [startSites\(\)](#), [stopCodons\(\)](#), [stopSites\(\)](#), [txNames\(\)](#), [uniqueGroups\(\)](#), [uniqueOrder\(\)](#)

---

ORFik.template.experiment	<i>An ORFik experiment to see how it looks</i>
---------------------------	--

---

**Description**

NOTE! This experiment should only be used for testing, since it is just sampled data internal in ORFik.

**Usage**

```
ORFik.template.experiment(as.temp = FALSE)
```

**Arguments**

as.temp	logical, default FALSE, load as ORFik experiment. If TRUE, loads as data.frame template of the experiment.
---------	--



**Value**

an ORFik [experiment](#)

**See Also**

Other ORFik\_experiment: [bamVarName\(\)](#), [create.experiment\(\)](#), [experiment-class](#), [filepath\(\)](#), [libraryTypes\(\)](#), [organism.df\(\)](#), [outputLibs\(\)](#), [read.experiment\(\)](#), [save.experiment\(\)](#), [validateExperiments\(\)](#)

**Examples**

```
ORFik.template.experiment()
```

---

 ORFikQC

*A post Alignment quality control of reads*


---

**Description**

The ORFik QC uses the aligned files (usually bam files), fastp and STAR log files combined with annotation to create relevant statistics.

This report consists of several steps:

1. From this report you will get a summary csv table, with distribution of aligned reads and overlap counts over transcript regions like: leader, cds, trailer, lincRNAs, tRNAs, rRNAs, snoRNAs etc. It will be called STATS.csv. And can be imported with [QCstats](#) function.
2. It will also make correlation plots and meta coverage plots, so you get a good understanding of how good the quality of your NGS data production + aligner step were.
3. Count tables are produced, similar to HTseq count tables. Over mrna, leader, cds and trailer separately. This tables are stored as [SummarizedExperiment](#), for easy loading into DEseq, conversion to normalized fpkm values, or collapsing replicates in an experiment. And can be imported with [countTable](#) function.

Everything will be outputted in the directory of your NGS data, inside the folder `./QC_STATS/`, relative to data location in `'df'`. You can specify new out location with `out.dir` if you want.

To make a ORFik experiment, see `?ORFik::experiment`

To see some normal mrna coverage profiles of different RNA-seq protocols: <https://www.ncbi.nlm.nih.gov/pmc/articles/P>

**Usage**

```
ORFikQC(df, out.dir = dirname(df$filepath[1]), BPPARAM = bpparam())
```

**Arguments**

<code>df</code>	an ORFik <a href="#">experiment</a>
<code>out.dir</code>	optional output directory, default: <code>dirname(df\$filepath[1])</code> . Will make a folder called "QC_STATS" with all results in this directory.
<code>BPPARAM</code>	how many cores/threads to use? default: <code>bpparam()</code> . To see number of threads used, do <code>bpparam()\$workers</code>

**Value**

`invisible(NULL)` (objects are stored to disc)

**See Also**

Other QC report: [QCplots\(\)](#), [QCstats\(\)](#)

**Examples**

```
# Load an experiment
df <- ORFik.template.experiment()
# Run QC
# QCreport(df)
```

---

 orfScore

*Get ORFscore for a GRangesList of ORFs*


---

**Description**

ORFscore tries to check whether the first frame of the 3 possible frames in an ORF has more reads than second and third frame. **IMPORTANT:** Only use p-shifted libraries, see ([detectRibosomeShifts](#)). Else this score makes no sense.

**Usage**

```
orfScore(gr1, RFP, is.sorted = FALSE, weight = "score", overlapGr1 = NULL)
```

**Arguments**

gr1	a <a href="#">GRangesList</a> of 5' utrs, CDS, transcripts, etc.
RFP	ribosomal footprints, given as <a href="#">GAlignments</a> or <a href="#">GRanges</a> object, must be already shifted and resized to the p-site. Requires a \$size column with original read lengths.
is.sorted	logical (FALSE), is gr1 sorted. That is + strand groups in increasing ranges (1,2,3), and - strand groups in decreasing ranges (3,2,1)
weight	(default: 'score'), if defined a character name of valid meta column in subject. <a href="#">GRanges("chr1", 1, "+", score = 5)</a> , would mean score column tells that this alignment region was found 5 times. ORFik .bedo files, contains a score column like this. As do CAGEr CAGE files and many other package formats. You can also assign a score column manually.
overlapGr1	an integer, (default: NULL), if defined must be <a href="#">countOverlaps(gr1, RFP)</a> , added for speed if you already have it

**Details**

Pseudocode: assume rff - is reads fraction in specific frame

$$\text{ORFscore} = \log(\text{rrf1} + \text{rrf2} + \text{rrf3})$$

If rrf2 or rrf3 is bigger than rff1, negate the resulting value.

$$\text{ORFscore}[\text{rrf1Smaller}] <- \text{ORFscore}[\text{rrf1Smaller}] * -1$$

As result there is one value per ORF: Positive values say that the first frame have the most reads, negative values say that the first frame does not have the most reads. NOTE: If reads are not of width 1, then a read from 1-4 on range of 1-4, will get scores frame1 = 2, frame2 = 1, frame3 = 1. What could be logical is that only the 5' end is important, so that only frame1 = 1, to get this, you first resize reads to 5' end only.

NOTES: 1. p shifting is not exact, so some functional ORFs will get a bad ORF score.

2. If a score column is defined, it will use it as weights, set to weight = 1L if you don't have weight, and score column is something else. see [getWeights](#)

## Value

a data.table with 4 columns, the orfscore (ORFScores) and score of each of the 3 tiles (frame\_zero\_RP, frame\_one\_RP, frame\_two\_RP)

## References

doi: 10.1002/embj.201488411

## See Also

Other features: [computeFeaturesCage\(\)](#), [computeFeatures\(\)](#), [countOverlapsW\(\)](#), [disengagementScore\(\)](#), [distToCds\(\)](#), [distToTSS\(\)](#), [entropy\(\)](#), [floss\(\)](#), [fpkm\\_calc\(\)](#), [fpkm\(\)](#), [fractionLength\(\)](#), [initiationScore\(\)](#), [insideOutsideORF\(\)](#), [isInFrame\(\)](#), [isOverlapping\(\)](#), [kozakSequenceScore\(\)](#), [rankOrder\(\)](#), [ribosomeReleaseScore\(\)](#), [ribosomeStallingScore\(\)](#), [startRegionCoverage\(\)](#), [startRegion\(\)](#), [stopRegion\(\)](#), [subsetCoverage\(\)](#), [translationalEff\(\)](#)

## Examples

```
ORF <- GRanges(seqnames = "1",
               ranges = IRanges(start = c(1, 10, 20), end = c(5, 15, 25)),
               strand = "+")
names(ORF) <- c("tx1", "tx1", "tx1")
gr1 <- GRangesList(tx1_1 = ORF)
RFP <- GRanges("1", IRanges(25, 25), "+") # 1 width position based
score(RFP) <- 28 # original width
orfScore(gr1, RFP) # negative because more hits on frames 1,2 than 0.

# example with positive result, more hits on frame 0 (in frame of ORF)
RFP <- GRanges("1", IRanges(c(1, 1, 1, 25), width = 1), "+")
score(RFP) <- c(28, 29, 31, 28) # original width
orfScore(gr1, RFP)
```

---

organism.df

*Get organism of the ORFik experiment*

---

## Description

Uses the txdb / gtf organism information, if existing.

## Usage

```
organism.df(df)
```

**Arguments**

df                    an ORFik [experiment](#)

**Value**

organism (character vector), if no organism set: NA

**See Also**

Other ORFik\_experiment: [ORFik.template.experiment\(\)](#), [bamVarName\(\)](#), [create.experiment\(\)](#), [experiment-class](#), [filepath\(\)](#), [libraryTypes\(\)](#), [outputLibs\(\)](#), [read.experiment\(\)](#), [save.experiment\(\)](#), [validateExperiments\(\)](#)

**Examples**

```
# if you have set organism in txdb of
# ORFik experiment:
df <- ORFik.template.experiment()
#organism.df(df)

#' If you have not set the organism you can do:
#txdb <- GenomicFeatures::makeTxDbFromGFF("pat/to/gff_or_gff")
#BiocGenerics::organism(txdb) <- "Homo sapiens"
#saveDb(txdb, paste0("pat/to/gff_or_gff", ".db"))
# then use this txdb in you ORFik experiment and load:
# create.experiment(exper = "new_experiment",
#   txdb = paste0("pat/to/gff_or_gff", ".db")) ...
# organism.df(read.experiment("new-experiment"))
```

---

outputLibs

*Output bam/bed/bedo/bedoc/ofst/wig files to R as variables*

---

**Description**

Variable names defined by df (ORFik experiment DataFrame) Uses multiple cores to load, defined by multicoreParam

**Usage**

```
outputLibs(
  df,
  chrStyle = NULL,
  type = "default",
  envir = .GlobalEnv,
  BPPARAM = bpparam()
)
```

**Arguments**

df	an ORFik <a href="#">experiment</a>
chrStyle	a GRanges object, TxDb, FaFile, or a <a href="#">seqlevelsStyle</a> (Default: NULL) to get seqlevelsStyle from. Is chromosome 1 called chr1 or 1, is mitochondrial chromosome called MT or chrM etc. Will use 1st seqlevel-style if more are present. Like: c("NCBI", "UCSC") -> pick "NCBI"
type	a character(default: "default"), load files in experiment or some precomputed variant, either "bedo", "bedoc", "ofst" or "pshifted". These are made with ORFik::simpleLibs(), shiftFootprintsByExperiment()..
envir	environment to save to, default (.GlobalEnv)
BPPARAM	how many cores/threads to use? default: bpparam(). To see number of threads used, do bpparam()\$workers

**Value**

NULL (libraries set by envir assignment)

**See Also**

Other ORFik\_experiment: [ORFik.template.experiment\(\)](#), [bamVarName\(\)](#), [create.experiment\(\)](#), [experiment-class](#), [filepath\(\)](#), [libraryTypes\(\)](#), [organism.df\(\)](#), [read.experiment\(\)](#), [save.experiment\(\)](#), [validateExperiments\(\)](#)

**Examples**

```
## Load a template ORFik experiment
df <- ORFik.template.experiment()
## Default library type load, usually bam files
# outputLibs(df, type = "default")
## .ofst file load, if ofst files does not exists
## it will load default
# outputLibs(df, type = "ofst")
## .wig file load, if wiggle files does not exists
## it will load default
# outputLibs(df, type = "wig")
```

---

pasteDir	<i>A paste function for directories Makes sure slashes are corrected, and not doubled.</i>
----------	--

---

**Description**

A paste function for directories Makes sure slashes are corrected, and not doubled.

**Usage**

```
pasteDir(...)
```

**Arguments**

... any amount of arguments that are possible to convert to characters

**Value**

the pasted string

---

percentage\_to\_ratio     *Convert percentage to ratio of 1*

---

**Description**

50 -> 0.5 etc, if length cds > minimum.cds

**Usage**

```
percentage_to_ratio(top_tx, cds, minimum.cds = 1000)
```

**Arguments**

top_tx	numeric
cds	GRangesList object
minimum.cds	numeric, default 1000

**Value**

numeric, as ratio of 1

---

plotHelper     *Helper function for coverage plots*

---

**Description**

Should only be used internally

**Usage**

```
plotHelper(
  coverage,
  df,
  outdir,
  scores,
  returnCoverage = FALSE,
  title = "coverage metaplot",
  colors = c("skyblue4", "orange"),
  plotFunction = "windowCoveragePlot"
)
```

**Arguments**

coverage	a data.table containing at least columns (count/score, position), it is possible to have additional: (genes, fraction, feature)
df	an ORFik <a href="#">experiment</a>
outdir	directory to save to (default: NULL, no saving)
scores	scoring function (default: c("sum", "zscore")), see ?coverageScorings for possible scores.
returnCoverage	(default: FALSE), return the ggplot object (TRUE) or NULL (FALSE).
title	Title to give plot
colors	Which colors to use, default auto color from function <a href="#">experiment.colors</a> , new color per library type. Else assign colors yourself.
plotFunction	Which plot function, default: windowCoveragePlot

**Value**

NULL (or ggplot object if returnCoverage is TRUE)

---

pmapFromTranscriptF     *Faster pmapFromTranscript*

---

**Description**

Map range coordinates between features in the transcriptome and genome (reference) space. The length of x must be the same as length of transcripts. Only exception is if x have integer names like (1, 3, 3, 5), so that x[1] maps to 1, x[2] maps to transcript 3 etc.

**Usage**

```
pmapFromTranscriptF(x, transcripts, removeEmpty = FALSE)
```

**Arguments**

x	IRangesList/IRanges/GRanges to map to genomic coordinates
transcripts	a GRangesList to map against (the genomic coordinates)
removeEmpty	a logical, remove non hit exons, else they are set to 0. That is all exons in the reference that the transcript coordinates do not span.

**Details**

This version tries to fix the short comings of GenomicFeature's version. Much faster and uses less memory. Implemented as dynamic program optimized c++ code.

**Value**

a GRangesList of mapped reads, names from ranges are kept.

**Examples**

```

ranges <- IRanges(start = c( 5, 6), end = c(10, 10))
seqnames = rep("chr1", 2)
strands = rep("-", 2)
gr1 <- split(GRanges(seqnames, IRanges(c(85, 70), c(89, 82)), strands),
            c(1, 1))
ranges <- split(ranges, c(1,1)) # both should be mapped to transcript 1
pmapFromTranscriptF(ranges, gr1, TRUE)

```

---

pmapToTranscriptF      *Faster pmapToTranscript*

---

**Description**

Map range coordinates between features in the transcriptome and genome (reference) space. The length of x must be the same as length of transcripts. Only exception is if x have integer names like (1, 3, 3, 5), so that x[1] maps to 1, x[2] maps to transcript 3 etc.

**Usage**

```

pmapToTranscriptF(
  x,
  transcripts,
  ignore.strand = FALSE,
  x.is.sorted = TRUE,
  tx.is.sorted = TRUE
)

```

**Arguments**

x	GRangesList/GRanges/IRangesList/IRanges to map to transcriptomic coordinates
transcripts	a GRangesList/GRanges/IRangesList/IRanges to map against (the genomic coordinates). Must be of lower abstraction level than x. So if x is GRanges, transcripts can not be IRanges etc.
ignore.strand	When ignore.strand is TRUE, strand is ignored in overlaps operations (i.e., all strands are considered "+") and the strand in the output is '*'. When ignore.strand is FALSE (default) strand in the output is taken from the transcripts argument. When transcripts is a GRangesList, all inner list elements of a common list element must have the same strand or an error is thrown. Mapped position is computed by counting from the transcription start site (TSS) and is not affected by the value of ignore.strand.
x.is.sorted	if x is a GRangesList object, are "-" strand groups pre-sorted in decreasing order within group, default: TRUE
tx.is.sorted	if transcripts is a GRangesList object, are "-" strand groups pre-sorted in decreasing order within group, default: TRUE

**Details**

This version tries to fix the shortcomings of GenomicFeature's version. Much faster and uses less memory. Implemented as dynamic program optimized c++ code.



**Value**

object of same class as input x, names from ranges are kept.

**Examples**

```

ranges <- IRanges(start = c(5, 6), end = c(10, 10))
seqnames = rep("chr1", 2)
strands = rep("-", 2)
gr1 <- split(GRanges(seqnames, IRanges(c(85, 70), c(89, 82)), strands),
            c(1, 1))
ranges <- split(ranges, c(1,1)) # both should be mapped to transcript 1
pmapFromTranscriptF(ranges, gr1, TRUE)

```

---

prettyScoring	<i>Prettify scoring name</i>
---------------	------------------------------

---

**Description**

Prettify scoring name

**Usage**

```
prettyScoring(scoring)
```

**Arguments**

scoring            a character (the scoring)

**Value**

a new scoring name or the same if pretty

---

pseudo.transform	<i>Transform object</i>
------------------	-------------------------

---

**Description**

Similar to normal transform like log2 or log10. But keep 0 values as 0, to avoid Inf values and negative values are made as -scale(abs(x)), to avoid NaN values.

**Usage**

```
pseudo.transform(x, scale = log2, by.reference = FALSE)
```

**Arguments**

x                    a numeric vector or data.frame/data.table of numeric columns  
scale                a function, default log2, which function to transform with.  
by.reference        logical, FALSE. if TRUE, update object by reference if it is data.table.

**Value**

same object class as x, with transformed values

---

pSitePlot

*Plot area around TIS as histogram*

---

**Description**

Usefull to validate p-shifting is correct Can be used for any coverage of region around a point, like TIS, TSS, stop site etc.

**Usage**

```
pSitePlot(
  hitMap,
  length = 29,
  region = "start",
  output = NULL,
  type = "canonical CDS",
  scoring = "Averaged counts",
  forHeatmap = FALSE
)
```

**Arguments**

hitMap	a data.frame/data.table, given from metaWindow (must have columns: position, (score or count) and frame)
length	an integer (29), which length is this for?
region	a character (start), either "start or "stop"
output	character (NULL), if set, saves the plot as pdf or png to path given. If no format is given, is save as pdf.
type	character (canonical CDS), type for plot
scoring	character, default: (Averaged counts), which scoring did you use ? see ?coverageScorings for info and more alternatives.
forHeatmap	a logical (FALSE), should the plot be part of a heatmap? It will scale it differently. Removing title, x and y labels, and truncate spaces between bars.

**Details**

The region is represented as a histogram with different colors for the 3 frames. To make it easy to see patterns in the reads. Remember if you want to change anything like colors, just return the ggplot object, and reassign like: obj + scale\_color\_brewer() etc.

**Value**

a ggplot object of the coverage plot, NULL if output is set, then the plot will only be saved to location.

**See Also**

Other coveragePlot: [coverageHeatMap\(\)](#), [savePlot\(\)](#), [windowCoveragePlot\(\)](#)

**Examples**

```
# An ORF
gr1 <- GRangesList(tx1 = GRanges("1", IRanges(1, 6), "+"))
# Ribo-seq reads
range <- IRanges(c(rep(1, 3), 2, 3, rep(4, 2), 5, 6), width = 1)
reads <- GRanges("1", range, "+")
coverage <- coveragePerTiling(gr1, reads, TRUE, as.data.table = TRUE,
                             withFrames = TRUE)

pSitePlot(coverage)

# See vignette for more examples
```

QCplots

*Correlation and coverage plots for ORFikQC***Description**

Correlation plots default to mRNA covering reads. Meta plots defaults to leader, cds, trailer. Output will be stored in same folder as the libraries in df. Correlation plots will be fpkm correlation and  $\log_2(\text{fpkm} + 1)$  correlation between samples.

**Usage**

```
QCplots(
  df,
  region = "mrna",
  stats_folder = paste0(dirname(df$filepath[1]), "/QC_STATS/")
)
```

**Arguments**

df	an ORFik <a href="#">experiment</a>
region	a character (default: mrna), make raw count matrices of whole mrnas or one of (leaders, cds, trailers)
stats_folder	directory to save, default: <code>paste0(dirname(df\$filepath[1]), "/QC_STATS/")</code>

**Details**

Is part of [QCreport](#)

**Value**

invisible(NULL) (objects stored to disc)

**See Also**

Other QC report: [QCreport\(\)](#), [QCstats\(\)](#)

**Description**

The ORFik QC uses the aligned files (usually bam files), fastp and STAR log files combined with annotation to create relevant statistics.

This report consists of several steps:

1. From this report you will get a summary csv table, with distribution of aligned reads and overlap counts over transcript regions like: leader, cds, trailer, lincRNAs, tRNAs, rRNAs, snoRNAs etc. It will be called STATS.csv. And can be imported with [QCstats](#) function.
2. It will also make correlation plots and meta coverage plots, so you get a good understanding of how good the quality of your NGS data production + aligner step were.
3. Count tables are produced, similar to HTseq count tables. Over mrna, leader, cds and trailer separately. This tables are stored as [SummarizedExperiment](#), for easy loading into DEseq, conversion to normalized fpkm values, or collapsing replicates in an experiment. And can be imported with [countTable](#) function.

Everything will be outputed in the directory of your NGS data, inside the folder `./QC_STATS/`, relative to data location in 'df'. You can specify new out location with `out.dir` if you want.

To make a ORFik experiment, see `?ORFik::experiment`

To see some normal mrna coverage profiles of different RNA-seq protocols: <https://www.ncbi.nlm.nih.gov/pmc/articles/P>

**Usage**

```
QCreport(df, out.dir = dirname(df$filepath[1]), BPPARAM = bpparam())
```

**Arguments**

df	an ORFik <a href="#">experiment</a>
out.dir	optional output directory, default: <code>dirname(df\$filepath[1])</code> . Will make a folder called "QC_STATS" with all results in this directory.
BPPARAM	how many cores/threads to use? default: <code>bpparam()</code> . To see number of threads used, do <code>bpparam()\$workers</code>

**Value**

invisible(NULL) (objects are stored to disc)

**See Also**

Other QC report: [QCplots\(\)](#), [QCstats\(\)](#)

**Examples**

```
# Load an experiment
df <- ORFik.template.experiment()
# Run QC
# QCreport(df)
```

---

QCstats                      *Load ORFik QC Statistics report*

---

### Description

Loads the pre / post alignment statistics made in ORFik.

### Usage

```
QCstats(df, path = file.path(dirname(df$filepath[1]), "/QC_STATS/STATS.csv"))
```

### Arguments

df	an ORFik <a href="#">experiment</a>
path	path to QC statistics report, default: file.path(dirname(df\$filepath[1]), "/QC_STATS/STATS.csv")

### Details

The ORFik QC uses the aligned files (usually bam files), fastp and STAR log files combined with annotation to create relevant statistics.

### Value

data.table of QC report or NULL if not exists

### See Also

Other QC report: [QCplots\(\)](#), [QCreport\(\)](#)

### Examples

```
df <- ORFik.template.experiment()[3,]
## First make QC report
# QCreport(df)
# stats <- QCstats(df)
```

---

QCstats.plot                      *Make plot of ORFik QCreport*

---

### Description

From post-alignment QC relative to annotation, make a plot for all samples. Will contain among others read lengths, reads overlapping leaders, cds, trailers, mRNA / rRNA etc.

### Usage

```
QCstats.plot(stats, output.dir = NULL)
```

**Arguments**

`stats` path to ORFik QC stats .csv file, or the experiment object.

`output.dir` NULL or character path, default: NULL, plot not saved to disc. If defined saves plot to that directory with the name "/STATS\_plot.png".

**Value**

ggplot object of the the statistics data

**Examples**

```
df <- ORFik.template.experiment()[3,]
## First make QC report
# QCreport(df)
## Now you can get plot
# QCstats.plot(df)
```

---

QC\_count\_tables

*Create count table info for QC report*

---

**Description**

The better the annotation / gtf used, the more results you get.

**Usage**

```
QC_count_tables(df, out.dir, BPPARAM = bpparam())
```

**Arguments**

`df` an ORFik [experiment](#)

`out.dir` optional output directory, default: `dirname(df$filepath[1])`. Will make a folder called "QC\_STATS" with all results in this directory.

`BPPARAM` how many cores/threads to use? default: `bpparam()`. To see number of threads used, do `bpparam()$workers`

**Value**

a data.table of the count info

---

rankOrder	<i>ORF rank in transcripts</i>
-----------	--------------------------------

---

### Description

Creates an ordering of ORFs per transcript, so that ORF with the most upstream start codon is 1, second most upstream start codon is 2, etc. Must input a grl made from ORFik, txNames\_2 -> 2.

### Usage

```
rankOrder(grl)
```

### Arguments

grl                    a [GRangesList](#) object with ORFs

### Value

a numeric vector of integers

### References

doi: 10.1074/jbc.R116.733899

### See Also

Other features: [computeFeaturesCage\(\)](#), [computeFeatures\(\)](#), [countOverlapsW\(\)](#), [disengagementScore\(\)](#), [distToCds\(\)](#), [distToTSS\(\)](#), [entropy\(\)](#), [floss\(\)](#), [fpkm\\_calc\(\)](#), [fpkm\(\)](#), [fractionLength\(\)](#), [initiationScore\(\)](#), [insideOutsideORF\(\)](#), [isInFrame\(\)](#), [isOverlapping\(\)](#), [kozakSequenceScore\(\)](#), [orfScore\(\)](#), [ribosomeReleaseScore\(\)](#), [ribosomeStallingScore\(\)](#), [startRegionCoverage\(\)](#), [startRegion\(\)](#), [stopRegion\(\)](#), [subsetCoverage\(\)](#), [translationalEff\(\)](#)

### Examples

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
                  ranges = IRanges(c(7, 14), width = 3),
                  strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
                  ranges = IRanges(c(4, 1), c(9, 3)),
                  strand = c("-", "-"))
grl <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
grl <- ORFik:::makeORFNames(grl)
rankOrder(grl)
```

---

read.experiment      *Read ORFik experiment*

---

### Description

Read in runs / samples from an experiment as a single R object. To read an ORFik experiment, you must of course make one first. See [create.experiment](#) The file must be csv and be a valid ORFik experiment

### Usage

```
read.experiment(file, in.dir = "~/Bio_data/ORFik_experiments/")
```

### Arguments

file	relative path to a ORFik experiment. That is a .csv file following ORFik experiment style ("," as seperator). , or a template data.frame from <a href="#">create.experiment</a> . Can also be full path to file, then in.dir argument is ignored.
in.dir	Directory to load experiment csv file from, default: "~/Bio_data/ORFik_experiments/" Set to NULL if you don't want to save it to disc. Does not apply if file is not a path, but a data.frame. Also does not apply if file was given as full path.

### Value

an ORFik [experiment](#)

### See Also

Other ORFik\_experiment: [ORFik.template.experiment\(\)](#), [bamVarName\(\)](#), [create.experiment\(\)](#), [experiment-class](#), [filepath\(\)](#), [libraryTypes\(\)](#), [organism.df\(\)](#), [outputLibs\(\)](#), [save.experiment\(\)](#), [validateExperiments\(\)](#)

### Examples

```
# From file
## Not run:
# Read from file
df <- read.experiment(filepath) # <- valid ORFik .csv file

## End(Not run)
## Read from (create.experiment() template)
df <- ORFik.template.experiment()

## To save it, do:
# save.experiment(df, file = "path/to/save/experiment")
## You can then do:
# read.experiment("path/to/save/experiment")
# or (identical):
# read.experiment("experiment", in.dir = "path/to/save/")
```



---

readBam	<i>Custom bam reader</i>
---------	--------------------------

---

### Description

Read in Bam file from either single end or paired end. Safer combined version of [readGAlignments](#) and [readGAlignmentPairs](#) that takes care of some common errors.

If QNAMES of the aligned reads are from collapsed fasta files, the bam file will contain a meta column called collapsed with the counts of duplicates per read.

### Usage

```
readBam(path, chrStyle = NULL, param = NULL)
```

### Arguments

path	a character path to .bam file. If paired end bam files, input must be a data.table with two columns (forward and reverse) and one row: if paired end reads in single bam file: forward contains paired end bam file path, reverse must be either "paired-end" or "" (single end). if paired end reads split in two files: forward contains paired end bam file path (R1), reverse must be paired end bam file path (R2 file), this is a rare case If all are single-end or you don't need to load data as paired end, the reverse column can be skipped.
chrStyle	a GRanges object, TxDb, FaFile, or a <a href="#">seqlevelsStyle</a> (Default: NULL) to get seqlevelsStyle from. Is chromosome 1 called chr1 or 1, is mitochondrial chromosome called MT or chrM etc. Will use 1st seqlevel-style if more are present. Like: c("NCBI", "UCSC") -> pick "NCBI"
param	NULL or a <a href="#">ScanBamParam</a> object. Like for <a href="#">scanBam</a> , this influences what fields and which records are imported. However, note that the fields specified thru this <a href="#">ScanBamParam</a> object will be loaded <i>in addition</i> to any field required for generating the returned object ( <a href="#">GAlignments</a> , <a href="#">GAlignmentPairs</a> , or <a href="#">GappedReads</a> object), but only the fields requested by the user will actually be kept as meta-data columns of the object.  By default (i.e. param=NULL or param=ScanBamParam()), no additional field is loaded. The flag used is scanBamFlag(isUnmappedQuery=FALSE) for <a href="#">readGAlignments</a> , <a href="#">readGAlignmentsList</a> , and <a href="#">readGappedReads</a> . (i.e. only records corresponding to mapped reads are loaded), and scanBamFlag(isUnmappedQuery=FALSE, isPaired=TRUE, has) for <a href="#">readGAlignmentPairs</a> (i.e. only records corresponding to paired-end reads with both ends mapped are loaded).

### Details

In the future will use a faster .bam loader for big .bam files in R.

### Value

a [GAlignments](#) or [GAlignmentPairs](#) object of bam file

**See Also**

Other utils: [bedToGR\(\)](#), [convertToOneBasedRanges\(\)](#), [export.bed12\(\)](#), [export.wiggle\(\)](#), [fimport\(\)](#), [findFa\(\)](#), [fread.bed\(\)](#), [optimizeReads\(\)](#), [readWig\(\)](#)

**Examples**

```
bam_file <- system.file("extdata", "ribo-seq.bam", package = "ORFik")
readBam(bam_file, "UCSC")
```

---

readLengthTable	<i>Make table of readlengths</i>
-----------------	----------------------------------

---

**Description**

Summarizing all libraries in experiment, make a table of proportion of read lengths.

**Usage**

```
readLengthTable(df, output.dir = NULL, type = "ofst", BPPARAM = bpparam())
```

**Arguments**

df	an ORFik <a href="#">experiment</a>
output.dir	NULL or character path, default: NULL, plot not saved to disc. If defined saves plot to that directory with the name <code>./readLengths.csv</code> .
type	character, default: "ofst". Type of library: either "default", usually bam format (the one you gave to experiment), "pshifted" pshifted reads, "ofst", "bed", "bedo" optimized bed, or "wig"
BPPARAM	how many cores/threads to use? default: <code>bpparam()</code> . To see number of threads used, do <code>bpparam()\$workers</code>

**Value**

a data.table object of the the read length data with columns: `c("sample", "sample_id", "read length", "counts", "counts_per_sample", "perc_of_counts_per_sample")`

---

readWidths	<i>Get read widths</i>
------------	------------------------

---

**Description**

Input any reads, e.g. ribo-seq object and get width of reads, this is to avoid confusion between width, qwidth and meta column containing original read width.

**Usage**

```
readWidths(reads, after.softclips = TRUE, along.reference = FALSE)
```

**Arguments**

`reads` a GRanges, GAlignment or GAlignmentPairs object.

`after.softclips` logical (TRUE), include softclips in width. Does not apply if `along.reference` is TRUE.

`along.reference` logical (FALSE), example: The cigar "26MI2" is by default width 28, but if `along.reference` is TRUE, it will be 26. The length of the read along the reference. Also "1D20M" will be 21 if by `along.reference` is TRUE. Intronic regions (cigar: N) will be removed. So: "1M200N19M" is 20, not 220.

**Details**

If input is p-shifted and GRanges, the "\$size" or "\$score" column must exist, and the column must contain the original read widths. In ORFik "\$size" have higher priority than "\$score" for defining length. ORFik P-shifting creates a \$size column, other softwares like shoelaces creates a score column.

Remember to think about how you define length. Like the question: is a Illumina error mismatch sufficient to reduce size of read and how do you know what is biological variance and what are Illumina errors?

**Value**

an integer vector of widths

**Examples**

```
gr <- GRanges("chr1", 1)
readWidths(gr)

# GAlignment with hit (1M) and soft clipped base (1S)
ga <- GAlignments(seqnames = "1", pos = as.integer(1), cigar = "1M1S",
  strand = factor("+", levels = c("+", "-", "*")))
readWidths(ga) # Without soft-clip bases

readWidths(ga, after.softclips = FALSE) # With soft-clip bases
```

---

readWig

*Custom wig reader*

---

**Description**

Given 2 wig files, first is forward second is reverse. Merge them and return as GRanges object. If they contain name reverse and forward, first and second order does not matter, it will search for forward and reverse.

**Usage**

```
readWig(path, chrStyle = NULL)
```

**Arguments**

path	a character path to two .wig files, or a data.table with 2 columns, (forward, filepath) and reverse, only 1 row.
chrStyle	a GRanges object, TxDb, FaFile, or a seqlevelsStyle (Default: NULL) to get seqlevelsStyle from. Is chromosome 1 called chr1 or 1, is mitochondrial chromosome called MT or chrM etc. Will use 1st seqlevel-style if more are present. Like: c("NCBI", "UCSC") -> pick "NCBI"

**Value**

a GRanges object of the file/s

**See Also**

Other utils: `bedToGR()`, `convertToOneBasedRanges()`, `export.bed12()`, `export.wiggle()`, `fimport()`, `findFa()`, `fread.bed()`, `optimizeReads()`, `readBam()`

---

reassignTSSbyCage	<i>Reassign all Transcript Start Sites (TSS)</i>
-------------------	--

---

**Description**

Given a GRangesList of 5' UTRs or transcripts, reassign the start sites using max peaks from CageSeq data. A max peak is defined as new TSS if it is within boundary of 5' leader range, specified by 'extension' in bp. A max peak must also be higher than minimum CageSeq peak cutoff specified in 'filterValue'. The new TSS will then be the positioned where the cage read (with highest read count in the interval). If removeUnused is TRUE, leaders without cage hits, will be removed, if FALSE the original TSS will be used.

**Usage**

```
reassignTSSbyCage(
  fiveUTRs,
  cage,
  extension = 1000,
  filterValue = 1,
  restrictUpstreamToTx = FALSE,
  removeUnused = FALSE,
  preCleanup = TRUE,
  cageMcol = FALSE
)
```

**Arguments**

fiveUTRs	(GRangesList) The 5' leaders or full transcript sequences
cage	Either a filePath for the CageSeq file as .bed .bam or .wig, with possible compressions (".gzip", ".gz", ".bgz"), or already loaded CageSeq peak data as GRanges or GAlignment. NOTE: If it is a .bam file, it will add a score column by running: <code>convertToOneBasedRanges(cage, method = "5prime", addScoreColumn = TRUE)</code> The score column is then number of replicates of read, if score column is something else, like read length, set the score column to NULL first.

extension	The maximum number of bases upstream of the TSS to search for CageSeq peak.
filterValue	The minimum number of reads on cage position, for it to be counted as possible new tss. (represented in score column in CageSeq data) If you already filtered, set it to 0.
restrictUpstreamToTx	a logical (FALSE). If TRUE: restrict leaders to not extend closer than 5 bases from closest upstream leader, set this to TRUE.
removeUnused	logical (FALSE), if False: (standard is to set them to original annotation), If TRUE: remove leaders that did not have any cage support.
preCleanup	logical (TRUE), if TRUE, remove all reads in region (-5:-1, 1:5) of all original tss in leaders. This is to keep original TSS if it is only +/- 5 bases from the original.
cageMcol	a logical (FALSE), if TRUE, add a meta column to the returned object with the raw CAGE counts in support for new TSS.

### Details

Note: If you used CAGER, you will get reads of a probability region, with always score of 1. Remember then to set filterValue to 0. And you should use the 5' end of the read as input, use: `ORFik:::convertToOneBasedRanges(cage)` NOTE on filtervalue: To get high quality TSS, set filtervalue to median count of reads overlapping per leader. This will make you discard a lot of new TSS positions though. I usually use 10 as a good standard.

TIP: do `summary(countOverlaps(fiveUTRs, cage))` so you can find a good cutoff value for noise.

### Value

a `GRangesList` of newly assigned TSS for fiveUTRs, using CageSeq data.

### See Also

Other CAGE: [assignTSSByCage\(\)](#), [reassignTxDbByCage\(\)](#)

### Examples

```
# example 5' leader, notice exon_rank column
fiveUTRs <- GenomicRanges::GRangesList(
  GenomicRanges::GRanges(seqnames = "chr1",
    ranges = IRanges::IRanges(1000, 2000),
    strand = "+",
    exon_rank = 1))
names(fiveUTRs) <- "tx1"

# make fake CageSeq data from promoter of 5' leaders, notice score column
cage <- GenomicRanges::GRanges(
  seqnames = "1",
  ranges = IRanges::IRanges(500, width = 1),
  strand = "+",
  score = 10) # <- Number of tags (reads) per position
# notice also that seqnames use different naming, this is fixed by ORFik
# finally reassign TSS for fiveUTRs
reassignTSSbyCage(fiveUTRs, cage)
# See vignette for example using gtf file and real CAGE data.
```

---

reassignTxDbByCage      *Input a txdb and reassign the TSS for each transcript by CAGE*

---

### Description

Given a TxDb object, reassign the start site per transcript using max peaks from CageSeq data. A max peak is defined as new TSS if it is within boundary of 5' leader range, specified by 'extension' in bp. A max peak must also be higher than minimum CageSeq peak cutoff specified in 'filterValue'. The new TSS will then be the positioned where the cage read (with highest read count in the interval).

### Usage

```
reassignTxDbByCage(
  txdb,
  cage,
  extension = 1000,
  filterValue = 1,
  restrictUpstreamToTx = FALSE,
  removeUnused = FALSE,
  preCleanup = TRUE
)
```

### Arguments

txdb	a TxDb file, a path to one of: (.gtf, .gff, .gff2, .gff2, .db or .sqlite) or an ORFik experiment
cage	Either a filePath for the CageSeq file as .bed .bam or .wig, with possible compressions (".gzip", ".gz", ".bgz"), or already loaded CageSeq peak data as GRanges or GAlignment. NOTE: If it is a .bam file, it will add a score column by running: convertToOneBasedRanges(cage, method = "5prime", addScoreColumn = TRUE) The score column is then number of replicates of read, if score column is something else, like read length, set the score column to NULL first.
extension	The maximum number of bases upstream of the TSS to search for CageSeq peak.
filterValue	The minimum number of reads on cage position, for it to be counted as possible new tss. (represented in score column in CageSeq data) If you already filtered, set it to 0.
restrictUpstreamToTx	a logical (FALSE). If TRUE: restrict leaders to not extend closer than 5 bases from closest upstream leader, set this to TRUE.
removeUnused	logical (FALSE), if False: (standard is to set them to original annotation), If TRUE: remove leaders that did not have any cage support.
preCleanup	logical (TRUE), if TRUE, remove all reads in region (-5:-1, 1:5) of all original tss in leaders. This is to keep original TSS if it is only +/- 5 bases from the original.

**Details**

Note: If you used CAGEr, you will get reads of a probability region, with always score of 1. Remember then to set filterValue to 0. And you should use the 5' end of the read as input, use: `ORFik:::convertToOneBasedRanges(cage)`

**Value**

a TxDb object of reassigned transcripts

**See Also**

Other CAGE: [assignTSSByCage\(\)](#), [reassignTSSbyCage\(\)](#)

**Examples**

```
## Not run:
library(GenomicFeatures)
# Get the gtf txdb file
txdbFile <- system.file("extdata", "hg19_knownGene_sample.sqlite",
  package = "GenomicFeatures")
cagePath <- system.file("extdata", "cage-seq-heart.bed.bgz",
  package = "ORFik")
reassignTxDbByCage(txdbFile, cagePath)

## End(Not run)
```

---

reduceKeepAttr	<i>Reduce GRanges / GRangesList</i>
----------------	-------------------------------------

---

**Description**

Reduce away all GRanges elements with 0-width.

**Usage**

```
reduceKeepAttr(
  grl,
  keep.names = FALSE,
  drop.empty.ranges = FALSE,
  min.gapwidth = 1L,
  with.revmap = FALSE,
  with.inframe.attrib = FALSE,
  ignore.strand = FALSE,
  min.strand.decreasing = TRUE
)
```

**Arguments**

`grl` a [GRangesList](#) or GRanges object

`keep.names` (FALSE) keep the names and meta columns of the GRangesList

`drop.empty.ranges` (FALSE) if a group is empty (width 0), delete it.

`min.gapwidth` (IL) how long gap can it be between two ranges, to merge them.  
`with.revmap` (FALSE) return info on which mapped to which  
`with.inframe.attrib`  
(FALSE) For internal use.  
`ignore.strand` (FALSE), can different strands be reduced together.  
`min.strand.decreasing`  
(TRUE), if GRangesList, return minus strand group ranges in decreasing order  
(1-5, 30-50) -> (30-50, 1-5)

### Details

Extends function `reduce` by trying to keep names and meta columns, if it is a GRangesList. It also does not lose sorting for GRangesList, since original reduce sorts all by ascending position. If `keep.names == FALSE`, it's just the normal `GenomicRanges::reduce` with sorting negative strands descending for GRangesList.

### Value

A reduced GRangesList

### See Also

Other `ExtendGenomicRanges`: `asTX()`, `coveragePerTiling()`, `extendLeaders()`, `extendTrailers()`, `tile1()`, `txSeqsFromFa()`, `windowPerGroup()`

### Examples

```

ORF <- GRanges(seqnames = "1",
               ranges = IRanges(start = c(1, 2, 3), end = c(1, 2, 3)),
               strand = "+")
# For GRanges
reduceKeepAttr(ORF, keep.names = TRUE)
# For GRangesList
grl <- GRangesList(tx1_1 = ORF)
reduceKeepAttr(grl, keep.names = TRUE)

```

---

`regionPerReadLength` *Find proportion of reads per position per read length in region*

---

### Description

This is defined as: Given some transcript region (like CDS), get coverage per position.

### Usage

```

regionPerReadLength(
  grl,
  reads,
  acceptedLengths = NULL,
  withFrames = TRUE,

```



```

    scoring = "transcriptNormalized",
    weight = "score",
    BPPARAM = bpparam()
  )

```

### Arguments

grl	a <a href="#">GRangesList</a> object with usually either leaders, cds', 3' utrs or ORFs
reads	a <a href="#">GAlignments</a> or <a href="#">GRanges</a> object of RiboSeq, RnaSeq etc. Weights for scoring is default the 'score' column in 'reads'
acceptedLengths	an integer vector (NULL), the read lengths accepted. Default NULL, means all lengths accepted.
withFrames	logical TRUE, add ORF frame (frame 0, 1, 2), starting on first position of every grl.
scoring	a character (transcriptNormalized), which meta coverage scoring ? one of (zscore, transcriptNormalized, mean, median, sum, sumLength, fracPos), see ?coverageScorings for more info. Use to decide a scoring of hits per position for metacoverage etc. Set to NULL if you do not want meta coverage, but instead want per gene per position raw counts.
weight	(default: 'score'), if defined a character name of valid meta column in subject. <a href="#">GRanges("chr1", 1, "+", score = 5)</a> , would mean score column tells that this alignment region was found 5 times. ORFik .bedo files, contains a score column like this. As do CAGEr CAGE files and many other package formats. You can also assign a score column manually.
BPPARAM	how many cores/threads to use? default: <code>bpparam()</code>

### Value

a data.table with lengths by coverage.

### See Also

Other coverage: [coverageScorings\(\)](#), [metaWindow\(\)](#), [scaledWindowPositions\(\)](#), [windowPerReadLength\(\)](#)

### Examples

```

# Raw counts per gene per position
cds <- GRangesList(tx1 = GRanges("1", 100:129, "+"))
reads <- GRanges("1", seq(79,129, 3), "+")
reads$size <- 28 # <- Set read length of reads
regionPerReadLength(cds, reads, scoring = NULL)
## Sum up reads in each frame per read length per gene
regionPerReadLength(cds, reads, scoring = "frameSumPerLG")

```

---

remakeTxdbExonIds	<i>Get new exon ids after update of txdb</i>
-------------------	--

---

**Description**

Get new exon ids after update of txdb

**Usage**

```
remakeTxdbExonIds(txList)
```

**Arguments**

txList            a list, call of as.list(txdb)

**Value**

a new valid ordered list of exon ids (integer)

---

remove.experiments	<i>Remove bam/bed/wig files load in R as variables</i>
--------------------	--

---

**Description**

Variable names defined by df, in envir defined

**Usage**

```
remove.experiments(df, envir = .GlobalEnv)
```

**Arguments**

df                an ORFik [experiment](#)  
envir             environment to save to, default (.GlobalEnv)

**Value**

NULL (objects removed from envir specified)

**Examples**

```
df <- ORFik.template.experiment()  
# Output to .GlobalEnv with:  
# outputLibs(df)  
# Then remove them with:  
# remove.experiments(df)
```

---

remove.file_ext	<i>Remove file extension of path</i>
-----------------	--------------------------------------

---

**Description**

Allows removal of compression

**Usage**

```
remove.file_ext(path, basename = FALSE)
```

**Arguments**

path	character path (allows multiple paths)
basename	relative path (TRUE) or full path (FALSE)? (default: FALSE)

**Value**

character path without file extension

---

removeMetaCols	<i>Removes meta columns</i>
----------------	-----------------------------

---

**Description**

Removes meta columns

**Usage**

```
removeMetaCols(gr1)
```

**Arguments**

gr1	a GRangesList or GRanges object
-----	---------------------------------

**Value**

same type and structure as input without meta columns

---

removeORFsWithinCDS     *Remove ORFs that are within cds*

---

**Description**

Remove ORFs that are within cds

**Usage**

```
removeORFsWithinCDS(gr1, cds)
```

**Arguments**

gr1                    (GRangesList), the ORFs to filter  
 cds                    (GRangesList), the coding sequences (main ORFs on transcripts), to filter against.

**Value**

(GRangesList) of filtered uORFs

**See Also**

Other uorfs: [addCdsOnLeaderEnds\(\)](#), [filterUORFs\(\)](#), [removeORFsWithSameStartAsCDS\(\)](#), [removeORFsWithSameStartAsCDS\(\)](#), [removeORFsWithStartInsideCDS\(\)](#), [uORFSearchSpace\(\)](#)

---

removeORFsWithSameStartAsCDS  
                                   *Remove ORFs that have same start site as the CDS*

---

**Description**

Remove ORFs that have same start site as the CDS

**Usage**

```
removeORFsWithSameStartAsCDS(gr1, cds)
```

**Arguments**

gr1                    (GRangesList), the ORFs to filter  
 cds                    (GRangesList), the coding sequences (main ORFs on transcripts), to filter against.

**Value**

(GRangesList) of filtered uORFs

**See Also**

Other uorfs: [addCdsOnLeaderEnds\(\)](#), [filterUORFs\(\)](#), [removeORFsWithSameStopAsCDS\(\)](#), [removeORFsWithStartInsideCDS\(\)](#), [removeORFsWithinCDS\(\)](#), [uORFSearchSpace\(\)](#)

---

`removeORFsWithSameStopAsCDS`*Remove ORFs that have same stop site as the CDS*

---

**Description**

Remove ORFs that have same stop site as the CDS

**Usage**

```
removeORFsWithSameStopAsCDS(gr1, cds)
```

**Arguments**

`gr1` (GRangesList), the ORFs to filter  
`cds` (GRangesList), the coding sequences (main ORFs on transcripts), to filter against.

**Value**

(GRangesList) of filtered uORFs

**See Also**

Other uorfs: [addCdsOnLeaderEnds\(\)](#), [filterUORFs\(\)](#), [removeORFsWithSameStartAsCDS\(\)](#), [removeORFsWithStartI](#)  
[removeORFsWithinCDS\(\)](#), [uORFSearchSpace\(\)](#)

---

`removeORFsWithStartInsideCDS`*Remove ORFs that have start site within the CDS*

---

**Description**

Remove ORFs that have start site within the CDS

**Usage**

```
removeORFsWithStartInsideCDS(gr1, cds)
```

**Arguments**

`gr1` (GRangesList), the ORFs to filter  
`cds` (GRangesList), the coding sequences (main ORFs on transcripts), to filter against.

**Value**

(GRangesList) of filtered uORFs

**See Also**

Other uorfs: [addCdsOnLeaderEnds\(\)](#), [filterUORFs\(\)](#), [removeORFsWithSameStartAsCDS\(\)](#), [removeORFsWithSameSt](#)  
[removeORFsWithinCDS\(\)](#), [uORFSearchSpace\(\)](#)

---

removeTxdbExons	<i>Remove exons in txList that are not in fiveUTRs</i>
-----------------	--

---

**Description**

Remove exons in txList that are not in fiveUTRs

**Usage**

```
removeTxdbExons(txList, fiveUTRs)
```

**Arguments**

txList	a list, call of as.list(txdb)
fiveUTRs	a GRangesList of 5' leaders

**Value**

a list, modified call of as.list(txdb)

---

removeTxdbTranscripts	<i>Remove specific transcripts in txdb List</i>
-----------------------	---

---

**Description**

Remove all transcripts, except the ones in fiveUTRs.

**Usage**

```
removeTxdbTranscripts(txList, fiveUTRs)
```

**Arguments**

txList	a list, call of as.list(txdb)
fiveUTRs	a GRangesList of 5' leaders

**Value**

a txList

---

rename.SRA.files	<i>Rename SRA files from metadata</i>
------------------	---------------------------------------

---

**Description**

Rename SRA files from metadata

**Usage**

```
rename.SRA.files(files, new_names)
```

**Arguments**

files	a character vector, with full path to all the files
new_names	a character vector of new names or a data.table with metadata to use to rename (usually from SRA metadata). Priority of renaming from the metadata is to check for unique names in the LibraryName column, then the sample_title column if no valid names in LibraryName. If found and still duplicates, will add "_rep1", "_rep2" to make them unique. Paired end data will get a extension of _p1 and _p2. If no valid names, will not rename, that is keep the SRR numbers, you then can manually rename files to something more meaningful.

**Value**

a character vector of new file names

---

repNames	<i>Get replicate name variants</i>
----------	------------------------------------

---

**Description**

Used to standardize nomenclature for experiments.

Example: 1 is main naming, but a variant is rep1 rep1 will then be renamed to 1

**Usage**

```
repNames()
```

**Value**

a data.table with 2 columns, the main name, and all name variants of the main name in second column as a list.

**See Also**

Other experiment\_naming: [cellLineNames\(\)](#), [conditionNames\(\)](#), [libNames\(\)](#), [mainNames\(\)](#), [stageNames\(\)](#), [tissueNames\(\)](#)

---

```
restrictTSSByUpstreamLeader
```

*Restrict extension of 5' UTRs to closest upstream leader end*

---

### Description

Basically this function restricts all startSites, to the upstream GRangesList objects end. Usually leaders, for CAGE. Example: leader1: start on 10, leader2: stop on 8, extend leader1 to 5 -> this function will resize leader1 to 9, to be outside leader2, so that CAGE reads can not wrongly overlap.

### Usage

```
restrictTSSByUpstreamLeader(fiveUTRs, shiftedfiveUTRs)
```

### Arguments

fiveUTRs            The 5' leader sequences as GRangesList

shiftedfiveUTRs

The 5' leader sequences as GRangesList shifted by CAGE

### Value

GRangesList object of restricted fiveUTRs

---

```
reverseMinusStrandPerGroup
```

*Reverse minus strand*

---

### Description

Reverse minus strand per group in a GRangesList Only reverse if minus strand is in increasing order

### Usage

```
reverseMinusStrandPerGroup(grl, onlyIfIncreasing = TRUE)
```

### Arguments

grl                    a [GRangesList](#)

onlyIfIncreasing

logical, default (TRUE), only reverse if decreasing

### Value

a [GRangesList](#)



---

RiboQC.plot	<i>Quality control for pshifted Ribo-seq data</i>
-------------	---

---

**Description**

Quality control for pshifted Ribo-seq data

**Usage**

```
RiboQC.plot(
  df,
  output.dir = file.path(dirname(df$filepath[1]), "QC_STATS/"),
  width = 6.6,
  height = 4.5,
  type = "pshifted",
  weight = "score",
  BPPARAM = bpparam()
)
```

**Arguments**

df	an ORFik <a href="#">experiment</a>
output.dir	directory to save plot, default: file.path(dirname(df\$filepath[1]), "QC_STATS/"). If NULL will not save.
width	width of plot, default 6.6 (in inches)
height	height of plot, default 4.5 (in inches)
type	type of library loaded, default pshifted, warning if not pshifted might crash if too many read lengths!
weight	which column in reads describe duplicates, default "score".
BPPARAM	how many cores/threads to use? default: bpparam(). To see number of threads used, do bpparam()\$workers

**Value**

ggplot object as a grid

---

ribosomeReleaseScore	<i>Ribosome Release Score (RRS)</i>
----------------------	-------------------------------------

---

**Description**

Ribosome Release Score is defined as

$$\frac{\text{(RPFs over ORF)}}{\text{(RPFs over 3' utrs)}}$$

and additionally normalized by lengths. If RNA is added as argument, it will normalize by RNA counts to justify location of 3' utrs. It can be understood as a ribosome stalling feature. A pseudo-count of one was added to both the ORF and downstream sums.

**Usage**

```
ribosomeReleaseScore(
  grl,
  RFP,
  GtfOrThreeUtrs,
  RNA = NULL,
  weight.RFP = 1L,
  weight.RNA = 1L,
  overlapGr1 = NULL
)
```

**Arguments**

<code>grl</code>	a <a href="#">GRangesList</a> object with usually either leaders, cds', 3' utrs or ORFs.
<code>RFP</code>	RiboSeq reads as <a href="#">GAlignments</a> , <a href="#">GRanges</a> or <a href="#">GRangesList</a> object
<code>GtfOrThreeUtrs</code>	if <code>Gtf</code> : a <a href="#">TxDb</a> object of a <code>gtf</code> file transcripts is called from: <code>'threeUTRsByTranscript(Gtf, use.names = TRUE)'</code> , if object is <a href="#">GRangesList</a> , it is presumed to be the 3' utrs
<code>RNA</code>	RnaSeq reads as <a href="#">GAlignments</a> , <a href="#">GRanges</a> or <a href="#">GRangesList</a> object
<code>weight.RFP</code>	a vector (default: 1L). Can also be character name of column in RFP. As in <code>translationalEff(weight = "score")</code> for: <code>GRanges("chr1", 1, "+", score = 5)</code> , would mean score column tells that this alignment region was found 5 times.
<code>weight.RNA</code>	Same as <code>weightRFP</code> but for RNA weights. (default: 1L)
<code>overlapGr1</code>	an integer, (default: NULL), if defined must be <code>countOverlaps(grl, RFP)</code> , added for speed if you already have it

**Value**

a named vector of numeric values of scores, NA means that no 3' utr was found for that transcript.

**References**

doi: 10.1016/j.cell.2013.06.009

**See Also**

Other features: [computeFeaturesCage\(\)](#), [computeFeatures\(\)](#), [countOverlapsW\(\)](#), [disengagementScore\(\)](#), [distToCds\(\)](#), [distToTSS\(\)](#), [entropy\(\)](#), [floss\(\)](#), [fpkm\\_calc\(\)](#), [fpkm\(\)](#), [fractionLength\(\)](#), [initiationScore\(\)](#), [insideOutsideORF\(\)](#), [isInFrame\(\)](#), [isOverlapping\(\)](#), [kozakSequenceScore\(\)](#), [orfScore\(\)](#), [rankOrder\(\)](#), [ribosomeStallingScore\(\)](#), [startRegionCoverage\(\)](#), [startRegion\(\)](#), [stopRegion\(\)](#), [subsetCoverage\(\)](#), [translationalEff\(\)](#)

**Examples**

```
ORF <- GRanges(seqnames = "1",
  ranges = IRanges(start = c(1, 10, 20), end = c(5, 15, 25)),
  strand = "+")
grl <- GRangesList(tx1_1 = ORF)
threeUTRs <- GRangesList(tx1 = GRanges("1", IRanges(40, 50), "+"))
RFP <- GRanges("1", IRanges(25, 25), "+")
RNA <- GRanges("1", IRanges(1, 50), "+")
ribosomeReleaseScore(grl, RFP, threeUTRs, RNA)
```

---

ribosomeStallingScore *Ribosome Stalling Score (RSS)*

---

### Description

Is defined as

$$\frac{\text{(RPFs over ORF stop sites)}}{\text{(RPFs over ORFs)}}$$

and normalized by lengths A pseudo-count of one was added to both the ORF and downstream sums.

### Usage

```
ribosomeStallingScore(grl, RFP, weight = 1L, overlapGr1 = NULL)
```

### Arguments

grl	a <a href="#">GRangesList</a> object with usually either leaders, cds', 3' utrs or ORFs.
RFP	RiboSeq reads as <a href="#">GAlignments</a> , <a href="#">GRanges</a> or <a href="#">GRangesList</a> object
weight	a vector (default: 1L, if 1L it is identical to <a href="#">countOverlaps()</a> ), if single number ( $\neq 1$ ), it applies for all, if more than one must be equal size of 'reads'. else it must be the string name of a defined meta column in subject "reads", that gives number of times a read was found. <a href="#">GRanges("chr1", 1, "+", score = 5)</a> , would mean "score" column tells that this alignment region was found 5 times.
overlapGr1	an integer, (default: NULL), if defined must be <a href="#">countOverlaps(grl, RFP)</a> , added for speed if you already have it

### Value

a named vector of numeric values of RSS scores

### References

doi: 10.1016/j.cels.2017.08.004

### See Also

Other features: [computeFeaturesCage\(\)](#), [computeFeatures\(\)](#), [countOverlapsW\(\)](#), [disengagementScore\(\)](#), [distToCds\(\)](#), [distToTSS\(\)](#), [entropy\(\)](#), [floss\(\)](#), [fpkm\\_calc\(\)](#), [fpkm\(\)](#), [fractionLength\(\)](#), [initiationScore\(\)](#), [insideOutsideORF\(\)](#), [isInFrame\(\)](#), [isOverlapping\(\)](#), [kozakSequenceScore\(\)](#), [orfScore\(\)](#), [rankOrder\(\)](#), [ribosomeReleaseScore\(\)](#), [startRegionCoverage\(\)](#), [startRegion\(\)](#), [stopRegion\(\)](#), [subsetCoverage\(\)](#), [translationalEff\(\)](#)

### Examples

```
ORF <- GRanges(seqnames = "1",
               ranges = IRanges(start = c(1, 10, 20), end = c(5, 15, 25)),
               strand = "+")
grl <- GRangesList(tx1_1 = ORF)
RFP <- GRanges("1", IRanges(25, 25), "+")
ribosomeStallingScore(grl, RFP)
```

---

rnaNormalize	<i>Normalize a data.table of coverage by RNA seq per position</i>
--------------	---

---

### Description

Normalizes per position per gene by this function: (reads at position / min(librarysize, 1) \* number of genes) / fpkm of that gene's RNA-seq

### Usage

```
rnaNormalize(coverage, df, dfr = NULL, tx, normalizeMode = "position")
```

### Arguments

coverage	a data.table containing at least columns (count/score, position), it is possible to have additional: (genes, fraction, feature)
df	an ORFik <a href="#">experiment</a>
dfr	an ORFik <a href="#">experiment</a> of RNA-seq to normalize against. Will add RNA normalized to plot name if this is done.
tx	a <a href="#">GRangesList</a> of mrna transcripts
normalizeMode	a character (default: "position"), how to normalize library against rna library. Either on "position", normalize by number of genes, sum of reads and RNA seq, on tx "region" or "feature": same as position but RNA is split into the feature groups to normalize. Useful if you have a list of targets and background genes.

### Details

Good way to compare libraries

### Value

a data.table of normalized transcripts by RNA.

---

save.experiment	<i>Save <a href="#">experiment</a> to disc</i>
-----------------	--

---

### Description

Save [experiment](#) to disc

### Usage

```
save.experiment(df, file)
```

### Arguments

df	an ORFik <a href="#">experiment</a>
file	name of file to save df as

**Value**

NULL (experiment save only)

**See Also**

Other ORFik\_experiment: [ORFik.template.experiment\(\)](#), [bamVarName\(\)](#), [create.experiment\(\)](#), [experiment-class](#), [filepath\(\)](#), [libraryTypes\(\)](#), [organism.df\(\)](#), [outputLibs\(\)](#), [read.experiment\(\)](#), [validateExperiments\(\)](#)

**Examples**

```
df <- ORFik.template.experiment()
## Save with:
#save.experiment(df, file = "path/to/save/experiment.csv")
## Identical (.csv not needed, can be added):
#save.experiment(df, file = "path/to/save/experiment")
```

---

 savePlot

*Helper function for writing plots to disc*


---

**Description**

Helper function for writing plots to disc

**Usage**

```
savePlot(plot, output = NULL, width = 200, height = 150, dpi = 300)
```

**Arguments**

plot	the ggplot to save
output	character string (NULL), if set, saves the plot as pdf or png to path given. If no format is given, is save as png.
width	width of output in mm
height	height of output in mm
dpi	(300) dpi of plot

**Value**

a ggplot object of the coverage plot, NULL if output is set, then the plot will only be saved to location.

**See Also**

Other coveragePlot: [coverageHeatMap\(\)](#), [pSitePlot\(\)](#), [windowCoveragePlot\(\)](#)

---

scaledWindowPositions *Scale (bin) windows to a meta window of given size*

---

### Description

For example scale a coverage table of a all human CDS to width 100

### Usage

```
scaledWindowPositions(
  grl,
  reads,
  scaleTo = 100,
  scoring = "meanPos",
  weight = "score",
  is.sorted = FALSE
)
```

### Arguments

grl	GRangesList or GRanges of your ranges
reads	GRanges object of your reads.
scaleTo	an integer (100), if windows have different size, a meta window can not directly be created, since a meta window must have equal size for all windows. Rescale all windows to scaleTo. i.e c(1,2,3) -> size 2 -> c(1, mean(2,3)) etc. Can also be a vector, 1 number per grl group.
scoring	a character, one of (meanPos, sumPos)
weight	(default: 'score'), if defined a character name of valid meta column in subject. GRanges("chr1", 1, "+", score = 5), would mean score column tells that this alignment region was found 5 times. ORFik .bedo files, contains a score column like this. As do CAGEr CAGE files and many other package formats. You can also assign a score column manually.
is.sorted	logical (FALSE), is grl sorted. That is + strand groups in increasing ranges (1,2,3), and - strand groups in decreasing ranges (3,2,1)

### Details

Nice for making metaplots, the score will be mean of merged positions.

### Value

A data.table with scored counts (counts) of reads mapped to positions (position) specified in windows along with frame (frame).

### See Also

Other coverage: [coverageScorings\(\)](#), [metaWindow\(\)](#), [regionPerReadLength\(\)](#), [windowPerReadLength\(\)](#)

**Examples**

```
library(GenomicRanges)
windows <- GRangesList(GRanges("chr1", IRanges(1, 200), "-"))
x <- GenomicRanges::GRanges(
  seqnames = "chr1",
  ranges = IRanges::IRanges(c(1, 100, 199), c(2, 101, 200)),
  strand = "-")
scaledWindowPositions(windows, x, scaleTo = 100)
```

---

scoreSummarizedExperiment

*Helper function for makeSummarizedExperimentFromBam*

---

**Description**

If txdb or gtf path is added, it is a rangedSummarizedExperiment For FPKM values, DESeq2::fpkm(robust = FALSE) is used

**Usage**

```
scoreSummarizedExperiment(
  final,
  score = "transcriptNormalized",
  collapse = FALSE
)
```

**Arguments**

final	ranged summarized experiment object
score	default: "transcriptNormalized" (row normalized raw counts matrix), alternative is "fpkm", "log2fpkm" or "log10fpkm"
collapse	a logical/character (default FALSE), if TRUE all samples within the group SAMPLE will be collapsed to one. If "all", all groups will be merged into 1 column called merged_all. Collapse is defined as $\text{rowSum}(\text{elements\_per\_group}) / \text{ncol}(\text{elements\_per\_group})$

**Value**

a DEseq summerizedExperiment object (transcriptNormalized) or matrix (if fpkm input)

---

seqnamesPerGroup	<i>Get list of seqnames per granges group</i>
------------------	---

---

**Description**

Get list of seqnames per granges group

**Usage**

```
seqnamesPerGroup(gr1, keep.names = TRUE)
```

**Arguments**

`gr1` a [GRangesList](#)  
`keep.names` a boolean, keep names or not, default: (TRUE)

**Value**

a character vector or Rle of seqnames(if seqnames == T)

**Examples**

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
                  ranges = IRanges(c(7, 14), width = 3),
                  strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
                   ranges = IRanges(c(4, 1), c(9, 3)),
                   strand = c("-", "-"))
gr1 <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
seqnamesPerGroup(gr1)
```

---

shiftFootprints	<i>Shift footprints by selected offsets</i>
-----------------	---

---

**Description**

Function shifts footprints (GRanges) using specified offsets for every of the specified lengths. Reads that do not conform to the specified lengths are filtered out and rejected. Reads are resized to single base in 5' end fashion, treated as p site. This function takes account for junctions in cigars of the reads. Length of the footprint is saved in size' parameter of GRanges output. Footprints are also sorted according to their genomic position, ready to be saved as a ofst, bed or wig file.

**Usage**

```
shiftFootprints(footprints, shifts, sort = TRUE)
```



**Arguments**

footprints	<a href="#">GAlignments</a> object of RiboSeq reads
shifts	a data.frame / data.table with minimum 2 columns, fraction (selected_lengths) and selected_shifts (relative position). Output from <a href="#">detectRibosomeShifts</a>
sort	logical, default TRUE. If False will keep original order of reads, and not sort output reads in increasing genomic location per chromosome and strand.

**Details**

The two columns in the shift data.frame/data.table argument are:

- fraction Numeric vector of lengths of footprints you select for shifting.
- offsets\_start Numeric vector of shifts for corresponding selected\_lengths. eg. c(-10, -10) with selected\_lengths of c(31, 32) means length of 31 will be shifted left by 10. Footprints of length 32 will be shifted right by 10.

NOTE: It will remove softclips from valid width, the CIGAR 3S30M is qwidth 33, but will remove 3S so final read width is 30 in ORFik.

**Value**

A [GRanges](#) object of shifted footprints, sorted and resized to 1bp of p-site, with metacolumn "size" indicating footprint size before shifting and resizing, sorted in increasing order.

**References**

<https://bmcbgenomics.biomedcentral.com/articles/10.1186/s12864-018-4912-6>

**See Also**

Other pshifting: [changePointAnalysis\(\)](#), [detectRibosomeShifts\(\)](#), [shiftFootprintsByExperiment\(\)](#)

**Examples**

```
# Basic run
#shiftFootprints(footprints, shifts)
# Full example
## Not run:
# input path to gtf, or load it as TxDb.
gtf_file <- system.file("extdata", "annotations.gtf", package = "ORFik")
# load reads
riboSeq_file <- system.file("extdata", "ribo-seq.bam", package = "ORFik")
footprints <- GenomicAlignments::readGAlignments(
  riboSeq_file, param = ScanBamParam(flag = scanBamFlag(
    isDuplicate = FALSE, isSecondaryAlignment = FALSE)))
# detect the shifts automagically
shifts <- detectRibosomeShifts(footprints, gtf_file)
# shift the RiboSeq footprints
shiftedReads <- shiftFootprints(footprints, shifts)

## End(Not run)
```

---

```
shiftFootprintsByExperiment
    Shift footprints of each file in experiment
```

---

## Description

For more details, see: [detectRibosomeShifts](#)

## Usage

```
shiftFootprintsByExperiment(
  df,
  out.dir = pasteDir(dirname(df$filepath[1]), "/pshifted/"),
  start = TRUE,
  stop = FALSE,
  top_tx = 10L,
  minFiveUTR = 30L,
  minCDS = 150L,
  minThreeUTR = 30L,
  firstN = 150L,
  min_reads = 1000,
  accepted.lengths = 26:34,
  output_format = c("ofst", "wig"),
  BPPARAM = bpparam(),
  log = TRUE,
  heatmap = FALSE,
  must.be.periodic = TRUE
)
```

## Arguments

df	an ORFik <a href="#">experiment</a>
out.dir	output directory for files, default: <code>dirname(df\$filepath[1])</code> , making a <code>/pshifted</code> folder at that location
start	(logical) Whether to include predictions based on the start codons. Default TRUE.
stop	(logical) Whether to include predictions based on the stop codons. Default FALSE. Only use if there exists 3' UTRs for the annotation. If periodicity around stop codon is stronger than at the start codon, use stop instead of start region for p-shifting.
top_tx	(integer), default 10. Specify which reads transcripts to use for estimation of the shifts. By default we take top 10 top covered transcripts as they represent less noisy dataset. This is only applicable when there are more than 1000 transcripts.
minFiveUTR	(integer) minimum bp for 5' UTR during filtering for the transcripts. Set to NULL if no 5' UTRs exists for annotation.
minCDS	(integer) minimum bp for CDS during filtering for the transcripts
minThreeUTR	(integer) minimum bp for 3' UTR during filtering for the transcripts. Set to NULL if no 3' UTRs exists for annotation.

firstN	(integer) Represents how many bases of the transcripts downstream of start codons to use for initial estimation of the periodicity.
min_reads	default (1000), how many reads must a read-length have to be considered for periodicity.
accepted.lengths	accepted readlengths, default 26:34, usually ribo-seq is strongest between 27:32.
output_format	default c("ofst", "wig"), use export.ofst or wiggle format (wig) using <a href="#">export.wiggle</a> ? Default is both. The wig format version can be used in IGV, the score column is counts of that read with that read length, the cigar reference width is lost, ofst is much faster to save and load in R, and retain cigar reference width, but can not be used in IGV. You can also do bedoc format, bed format keeping cigar: <a href="#">export.bedoc</a> . bedoc is usually not used for p-shifting.
BPPARAM	how many cores/threads to use? default: bpparam()
log	logical, default (TRUE), output a log file with parameters used.
heatmap	a logical or character string, default FALSE. If TRUE, will plot heatmap of raw reads before p-shifting to console, to see if shifts given make sense. You can also set a filepath to save the file there.
must.be.periodic	logical TRUE, if FALSE will not filter on periodic read lengths. (The Fourier transform filter will be skipped).

### Details

#' Saves files to a specified location as .ofst and .wig, The .ofst file will include a score column containing read width.  
The .wig files, will be saved in pairs of +/- strand, and score column will be replicates of reads starting at that position, score = 5 means 5 reads.  
Remember that different species might have different default Ribosome read lengths, for human, mouse etc, normally around 27:30.

### Value

NULL (Objects are saved to out.dir/pshited/"name\_pshifted.ofst", wig, bedo or .bedo)

### References

<https://bmcbgenomics.biomedcentral.com/articles/10.1186/s12864-018-4912-6>

### See Also

Other pshifting: [changePointAnalysis\(\)](#), [detectRibosomeShifts\(\)](#), [shiftFootprints\(\)](#)

### Examples

```
df <- ORFik.template.experiment()
df <- df[3,] #lets only p-shift RFP sample at index 3
# If you want to check it in IGV do:
shiftFootprintsByExperiment(df)
# Then use the .wig files that are created, which are readable in IGV.
# If you only need in R, do: (then you get no .wig files)
#shiftFootprintsByExperiment(df, output_format = "ofst")
```

---

 shiftPlots

*Plot shifted heatmaps per library*


---

## Description

A good validation for you p-shifting, to see shifts are corresponding and close to the CDS TIS.

## Usage

```
shiftPlots(
  df,
  output = NULL,
  title = "Ribo-seq",
  scoring = "transcriptNormalized",
  addFracPlot = TRUE,
  BPPARAM = bpparam()
)
```

## Arguments

df	an ORFik <a href="#">experiment</a>
output	name to save file, full path. (Default NULL) No saving.
title	Title for top of plot, default "Ribo-seq". A more informative name could be "Ribo-seq zebrafish Chew et al. 2013"
scoring	which scoring scheme to use for heatmap, default "transcriptNormalized". Some alternatives: "sum", "zscore".
addFracPlot	logical, default TRUE, add positional sum plot on top per heatmap.
BPPARAM	how many cores/threads to use? default: bpparam()

## Value

a ggplot2 grob object

## Examples

```
df <- ORFik.template.experiment()
df <- df[3,] #lets only p-shift RFP sample at index 3
#shiftFootprintsByExperiment(df, output_format = "bedo")
#shiftPlots(df, title = "Ribo-seq Human ORFik et al. 2020")
```

---

shifts.load	<i>Load the shifts from experiment</i>
-------------	--

---

### Description

When you p-shift using the function `shiftFootprintsByExperiment`, you will get a list of shifts per library. To automatically load them, you can use this function. Defaults to loading pshifts, if you made a-sites or e-sites, change the path argument to `ashifted/eshifted` folder instead.

### Usage

```
shifts.load(
  df,
  path = pasteDir(dirname(df$filepath[1]), "/pshifted/shifting_table.rds")
)
```

### Arguments

<code>df</code>	an ORFik <a href="#">experiment</a>
<code>path</code>	path to .rds file containing the shifts as a list, one list element per shifted bam file.

### Value

a list of the shifts, one list element per shifted bam file.

### Examples

```
df <- ORFik.template.experiment()
# subset on Ribo-seq
df <- df[df$libtype == "RFP",]
#shiftFootprintsByExperiment(df)
#shifts.load(df)
```

---

<code>show,experiment-method</code>	<i>experiment show definition</i>
-------------------------------------	-----------------------------------

---

### Description

Show a simplified version of experiment. The show function simplifies the view so that any column of data (like replicate or stage) is not shown, if all values are identical in that column. Filepath is also never shown.

### Usage

```
## S4 method for signature 'experiment'
show(object)
```

**Arguments**

object            an ORFik [experiment](#)

**Value**

print state of experiment

---

simpleLibs

*Converted format of NGS libraries*

---

**Description**

Export as either .ofst, .bedo or .bedoc files.

Export files as .bedo files: It is a bed file with 2 score columns. Gives a massive speedup when cigar string and bam flags are not needed.

Export files as .bedoc files: If cigar is needed, gives you replicates and cigar, so a fast way to load a GAlignment object, other bam flags are lost. If type is bedoc addSizeColumn and method will be ignored.

**Usage**

```
simpleLibs(
  df,
  out.dir = dirname(df$filepath[1]),
  addScoreColumn = TRUE,
  addSizeColumn = TRUE,
  must.overlap = NULL,
  method = "None",
  type = "ofst",
  reassign.when.saving = FALSE,
  envir = .GlobalEnv
)
```

**Arguments**

df                an ORFik [experiment](#)

out.dir           optional output directory, default: `dirname(df$filepath[1])`, if it is NULL, it will just reassign R objects to simplified libraries.

addScoreColumn   logical, default TRUE, if FALSE will not add replicate numbers as score column, see `ORFik::convertToOneBasedRanges`.

addSizeColumn    logical, default TRUE, if FALSE will not add size (width) as size column, see `ORFik::convertToOneBasedRanges`. Does not apply for .ofst or .bedoc.

must.overlap      default (NULL), else a GRanges / GRangesList object, so only reads that overlap (must.overlap) are kept. This is useful when you only need the reads over transcript annotation or subset etc.

method            character, default "None", the method to reduce ranges, for more info see [convertToOneBasedRanges](#)

type              a character of format, default "ofst". Alternatives: "ofst", "wig", "bedo" or "bedoc". Which format you want. Will make a folder within out.dir with this name containing the files.

reassign.when.saving      logical, default FALSE. If TRUE, will reassign library to converted form after saving. Ignored when out.dir = NULL.

envir                      which environment to save to, default .GlobalEnv

### Details

See [export.bedo](#) and [export.bedoc](#) for information on file formats

### Value

NULL (saves files to disc or R .GlobalEnv)

### Examples

```
df <- ORFik.template.experiment()
#convertLibs(df)
# Keep only 5' ends of reads
#convertLibs(df, method = "5prime")
```

---

sortPerGroup	<i>Sort a GRangesList</i>
--------------	---------------------------

---

### Description

A faster, more versatile reimplementaion of [sort.GenomicRanges](#) for GRangesList, needed since the original works poorly for more than 10k groups. This function sorts each group, where "+" strands are increasing by starts and "-" strands are decreasing by ends.

### Usage

```
sortPerGroup(grl, ignore.strand = FALSE, quick.rev = FALSE)
```

### Arguments

grl                      a [GRangesList](#)

ignore.strand      a boolean, (default FALSE): should minus strands be sorted from highest to lowest ends. If TRUE: from lowest to highest ends.

quick.rev            default: FALSE, if TRUE, given that you know all ranges are sorted from min to max for both strands, it will only reverse coordinates for minus strand groups, and only if they are in increasing order. Much quicker

### Details

Note: will not work if groups have equal names.

### Value

an equally named GRangesList, where each group is sorted within group.

**Examples**

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
  ranges = IRanges(c(14, 7), width = 3),
  strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
  ranges = IRanges(c(1, 4), c(3, 9)),
  strand = c("-", "-"))
grl <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
sortPerGroup(grl)
```

splitIn3Tx

*Create binned coverage of transcripts, split into the 3 parts.***Description**

The 3 parts of transcripts are the leaders, the cds' and trailers. Per transcript part, bin them all to windowSize (default 100), and make a data.table, rows are positions, useful for plotting with ORFik and ggplot2.

**Usage**

```
splitIn3Tx(
  leaders,
  cds,
  trailers,
  reads,
  windowSize = 100,
  fraction = "1",
  weight = "score",
  BPPARAM = bpparam()
)
```

**Arguments**

leaders	a <a href="#">GRangesList</a> of leaders (5' UTRs)
cds	a <a href="#">GRangesList</a> of coding sequences
trailers	a <a href="#">GRangesList</a> of trailers (3' UTRs)
reads	<a href="#">GRanges</a> or <a href="#">GAlignment</a> of reads
windowSize	an integer (100), size of windows (columns)
fraction	a character (1), info on reads (which read length, or which type (RNA seq)) (row names)
weight	(default: 'score'), if defined a character name of valid meta column in subject. <a href="#">GRanges</a> ("chr1", 1, "+", score = 5), would mean score column tells that this alignment region was found 5 times. ORFik .bedo files, contains a score column like this. As do CAGEr CAGE files and many other package formats. You can also assign a score column manually.
BPPARAM	how many cores/threads to use? default: <code>bpparam()</code>



**Value**

a data.table with columns position, score

---

stageNames	<i>Get stage name variants</i>
------------	--------------------------------

---

**Description**

Used to standardize nomenclature for experiments.

Example: 64Cell stage is same as 2 hours post fertilization, so all 2hpf will be converted to 64Cell etc.

**Usage**

```
stageNames()
```

**Value**

a data.table with 2 columns, the main name, and all name variants of the main name in second column as a list.

**See Also**

Other experiment\_naming: [cellLineNames\(\)](#), [conditionNames\(\)](#), [libNames\(\)](#), [mainNames\(\)](#), [repNames\(\)](#), [tissueNames\(\)](#)

---

STAR.align.folder	<i>Align all libraries in folder with STAR</i>
-------------------	--

---

**Description**

Does either all files as paired end or single end, so if you have mix, split them in two different folders.

If STAR halts at .... loading genome, it means the STAR index was aborted early, then you need to run: STAR.remove.crashed.genome(), with the genome that crashed, and rerun.

**Usage**

```
STAR.align.folder(
  input.dir,
  output.dir,
  index.dir,
  star.path = STAR.install(),
  fastp = install.fastp(),
  paired.end = FALSE,
  steps = "tr-ge",
  adapter.sequence = "auto",
  min.length = 20,
  mismatches = 3,
```

```

trim.front = 0,
alignment.type = "Local",
max.cpus = min(90, detectCores() - 1),
wait = TRUE,
include.subfolders = "n",
resume = NULL,
max.multimap = 10,
multiQC = TRUE,
script.folder = system.file("STAR_Aligner", "RNA_Align_pipeline_folder.sh", package =
  "ORFik"),
script.single = system.file("STAR_Aligner", "RNA_Align_pipeline.sh", package =
  "ORFik")
)

```

### Arguments

input.dir	path to fast files to align, the valid input files will be search for from formats: fast files (.fasta, .fastq, .fq, or.fa) with or without compression of .gz. Also either paired end or single end reads. Pairs will automatically be detected from similarity of naming, usually with a .1 and .2 in the end. If files are renamed, where pairs are not similarly named, this process will fail to find correct pairs.
output.dir	directory to save indices, default: paste0(dirname(arguments[1]), "/STAR_index/"), where arguments is the arguments input for this function.
index.dir	path to STAR index folder. Path returned from ORFik function STAR.index, when you created the index folders.
star.path	path to STAR, default: STAR.install(), if you don't have STAR installed at default location, it will install it there, set path to a runnable star if you already have it.
fastp	path to fastp trimmer, default: install.fastp(), if you have it somewhere else already installed, give the path. Only works for unix (linux or Mac OS), if not on unix, use your favorite trimmer and give the output files from that trimmer as input.dir here.
paired.end	a logical: default FALSE, alternative TRUE. If TRUE, will auto detect pairs by names. If yes running on a folder: The folder must then contain an even number of files and they must be named with the same prefix and suffix of either _1 and _2, 1 and 2, etc. If SRR numbers are used, it will start on lowest and match with second lowest etc.
steps	a character, default: "tr-ge", trimming then genome alignment steps of depletion and alignment wanted: The possible candidates you can use are: <ul style="list-style-type: none"> <li>• tr : trim reads</li> <li>• co : contamination merged depletion</li> <li>• ph : phix depletion</li> <li>• rR : rRNA depletion</li> <li>• nc : ncRNA depletion</li> <li>• tR : tRNA depletion</li> <li>• ge : genome alignment</li> <li>• all: run steps: "tr-co-ge" or "tr-ph-rR-nc-tR-ge", depending on if you have merged contaminants or not</li> </ul>

If not "all", a subset of these ("tr-co-ph-rR-nc-tR-ge")

If co (merged contaminants) is used, non of the specific contaminants can be specified, since they should be a subset of co.

The step where you align to the genome is usually always included, unless you are doing pure contaminant analysis. For Ribo-seq and TCP(RCP-seq) you should do rR (ribosomal RNA depletion), so when you made the STAR index you need the rRNA step, either use rRNA from .gtf or manual download. (usually just download a Silva rRNA database for SSU&LSU at: <https://www.arb-silva.de/>) for your species.

adapter.sequence	character, default: "auto". Auto detect adapter using fastp adapter auto detection, checking first 1.5M reads. (auto detect adapter, is not very reliable for Ribo-seq, so then you must include a manually specified, else alignment will most likely fail!). If already trimmed or trimming not wanted: adapter.sequence = "disable". You can manually assign adapter like: "ATCTCGTATGCCGTCTTCTGCTTG" or "AAAAAAAAAAAAA". You can also specify one of the three presets: <ul style="list-style-type: none"> <li>• illumina (standard for 100 bp sequencing): AGATCGGAAGAGC</li> <li>• small_RNA (standard for ~50 bp sequencing): TGGAATTCTCGG</li> <li>• nextera: CTGTCTCTTATA</li> </ul>
min.length	20, minimum length of aligned read without mismatches to pass filter.
mismatches	3, max non matched bases. Excludes soft-clipping, this only filters reads that have defined mismatches in STAR. Only applies for genome alignment step.
trim.front	0, default trim 0 bases 5'. For Ribo-seq set use 0. Ignored if tr (trim) is not one of the arguments in "steps"
alignment.type	default: "Local": standard local alignment with soft-clipping allowed, "EndToEnd" (global): force end-to-end read alignment, does not soft-clip.
max.cpus	integer, default: min(90, detectCores() - 1), number of threads to use. Default is minimum of 90 and maximum cores - 1. So if you have 8 cores it will use 7.
wait	a logical (not NA) indicating whether the R interpreter should wait for the command to finish, or run it asynchronously. This will be ignored (and the interpreter will always wait) if intern = TRUE. When running the command asynchronously, no output will be displayed on the Rgui console in Windows (it will be dropped, instead).
include.subfolders	"n" (no), do recursive search downwards for fast files if "y".
resume	default: NULL, continue from step, lets say steps are "tr-ph-ge": (trim, phix depletion, genome alignment) and resume is "ph", you will then use the assumed already trimmed data and start / continue from there starting at phix, usefull if something crashed. Like if you specified wrong STAR version, but the trimming step was completed.
max.multimap	numeric, default 10. If a read maps to more locations than specified, will skip the read. Set to 1 to only get unique mapping reads. Only applies for genome alignment step. The depletions are allowing for multimapping.
multiQC	logical, default TRUE. Do mutliQC comparison of STAR alignment between all the samples. Outputted in aligned/LOGS folder. See ?STAR.multiQC
script.folder	location of STAR index script, default internal ORFik file. You can change it and give your own if you need special alignments.

`script.single` location of STAR single file alignment script, default internal ORFik file. You can change it and give your own if you need special alignments.

### Details

Can only run on unix systems (Linux and Mac), and requires minimum 30GB memory on genomes like human, rat, zebrafish etc.

If for some reason the internal STAR alignment bash script will not work for you, like if you have a very small genome. You can copy the internal alignment script, edit it and give that as the Index script used for this function.

The trimmer used is fastp (the fastest I could find), works on mac and linux. If you want to use your own trimmer set file1/file2 to the location of the trimmed files from your program.

A note on trimming from creator of STAR about trimming: "adapter trimming it definitely needed for short RNA sequencing. For long RNA-seq, I would agree with Devon that in most cases adapter trimming is not advantageous, since, by default, STAR performs local (not end-to-end) alignment, i.e. it auto-trims." So trimming can be skipped for longer reads.

### Value

`output.dir`, can be used as as input in `ORFik::create.experiment`

### See Also

Other STAR: [STAR.align.single\(\)](#), [STAR.allsteps.multiQC\(\)](#), [STAR.index\(\)](#), [STAR.install\(\)](#), [STAR.multiQC\(\)](#), [STAR.remove.crashed.genome\(\)](#), [getGenomeAndAnnotation\(\)](#), [install.fastp\(\)](#)

### Examples

```
# First specify directories wanted
annotation.dir <- "~/Bio_data/references/Human"
fastq.input.dir <- "~/Bio_data/raw_data/Ribo_seq_subtelny/"
bam.output.dir <- "~/Bio_data/processed_data/Ribo_seq_subtelny_2014/"

## Download some SRA data and metadata
# info <- download.SRA.metadata("DRR041459", fastq.input.dir)
# download.SRA(info, fastq.input.dir, rename = FALSE)
## Now align 2 different ways, without and with contaminant depletion

## No contaminant depletion:
# annotation <- getGenomeAndAnnotation("Homo sapiens", annotation.dir)
# index <- STAR.index(annotation)
# STAR.align.folder(fastq.input.dir, bam.output.dir,
#                  index, paired.end = FALSE)

## All contaminants merged:
# annotation <- getGenomeAndAnnotation(
#   organism = "Homo_sapiens",
#   phix = TRUE, ncRNA = TRUE, tRNA = TRUE, rRNA = TRUE,
#   output.dir = annotation.dir
# )
# index <- STAR.index(annotation)
# STAR.align.folder(fastq.input.dir, bam.output.dir,
#                  index, paired.end = FALSE,
#                  steps = "tr-ge")
```

---

STAR.align.single      *Align single or paired end pair with STAR*

---

### Description

Given a single NGS fastq/fasta library, or a paired setup of 2 mated libraries. Run alignment and optionally remove contaminants.

### Usage

```
STAR.align.single(
  file1,
  file2 = NULL,
  output.dir,
  index.dir,
  star.path = STAR.install(),
  fastp = install.fastp(),
  steps = "tr-ge",
  adapter.sequence = "auto",
  min.length = 20,
  mismatches = 3,
  trim.front = 0,
  alignment.type = "Local",
  max.cpus = min(90, detectCores() - 1),
  wait = TRUE,
  resume = NULL,
  max.multimap = 10,
  script.single = system.file("STAR_Aligner", "RNA_Align_pipeline.sh", package =
    "ORFik")
)
```

### Arguments

file1	library file, if paired must be R1 file. Allowed formats are: (.fasta, .fastq, .fq, or .fa) with or without compression of .gz. This filename usually contains a suffix of .1
file2	default NULL, set if paired end to R2 file. Allowed formats are: (.fasta, .fastq, .fq, or .fa) with or without compression of .gz. This filename usually contains a suffix of .2
output.dir	directory to save indices, default: paste0(dirname(arguments[1]), "/STAR_index/"), where arguments is the arguments input for this function.
index.dir	path to STAR index folder. Path returned from ORFik function STAR.index, when you created the index folders.
star.path	path to STAR, default: STAR.install(), if you don't have STAR installed at default location, it will install it there, set path to a runnable star if you already have it.
fastp	path to fastp trimmer, default: install.fastp(), if you have it somewhere else already installed, give the path. Only works for unix (linux or Mac OS), if not on unix, use your favorite trimmer and give the output files from that trimmer as input.dir here.

steps	<p>a character, default: "tr-ge", trimming then genome alignment  steps of depletion and alignment wanted: The possible candidates you can use are:</p> <ul style="list-style-type: none"> <li>• tr : trim reads</li> <li>• co : contamination merged depletion</li> <li>• ph : phix depletion</li> <li>• rR : rRNA depletion</li> <li>• nc : ncRNA depletion</li> <li>• tR : tRNA depletion</li> <li>• ge : genome alignment</li> <li>• all: run steps: "tr-co-ge" or "tr-ph-rR-nc-tR-ge", depending on if you have merged contaminants or not</li> </ul> <p>If not "all", a subset of these ("tr-co-ph-rR-nc-tR-ge")  If co (merged contaminants) is used, none of the specific contaminants can be specified, since they should be a subset of co.  The step where you align to the genome is usually always included, unless you are doing pure contaminant analysis. For Ribo-seq and TCP(RCP-seq) you should do rR (ribosomal RNA depletion), so when you made the STAR index you need the rRNA step, either use rRNA from .gtf or manual download. (usually just download a Silva rRNA database for SSU&amp;LSU at: <a href="https://www.arb-silva.de/">https://www.arb-silva.de/</a>) for your species.</p>
adapter.sequence	<p>character, default: "auto". Auto detect adapter using fastp adapter auto detection, checking first 1.5M reads. (auto detect adapter, is not very reliable for Ribo-seq, so then you must include a manually specified, else alignment will most likely fail!). If already trimmed or trimming not wanted: adapter.sequence = "disable". You can manually assign adapter like: "ATCTCGTATGCCGTCTTCTGCTTG" or "AAAAAAAAAAAAA". You can also specify one of the three presets:</p> <ul style="list-style-type: none"> <li>• illumina (standard for 100 bp sequencing): AGATCGGAAGAGC</li> <li>• small_RNA (standard for ~50 bp sequencing): TGGAATTCTCGG</li> <li>• nextera: CTGTCTCTTATA</li> </ul>
min.length	20, minimum length of aligned read without mismatches to pass filter.
mismatches	3, max non matched bases. Excludes soft-clipping, this only filters reads that have defined mismatches in STAR. Only applies for genome alignment step.
trim.front	0, default trim 0 bases 5'. For Ribo-seq set use 0. Ignored if tr (trim) is not one of the arguments in "steps"
alignment.type	default: "Local": standard local alignment with soft-clipping allowed, "End-ToEnd" (global): force end-to-end read alignment, does not soft-clip.
max.cpus	integer, default: min(90, detectCores() - 1), number of threads to use. Default is minimum of 90 and maximum cores - 1. So if you have 8 cores it will use 7.
wait	a logical (not NA) indicating whether the R interpreter should wait for the command to finish, or run it asynchronously. This will be ignored (and the interpreter will always wait) if intern = TRUE. When running the command asynchronously, no output will be displayed on the Rgui console in Windows (it will be dropped, instead).

resume	default: NULL, continue from step, lets say steps are "tr-ph-ge": (trim, phix depletion, genome alignment) and resume is "ph", you will then use the assumed already trimmed data and start / continue from there starting at phix, usefull if something crashed. Like if you specified wrong STAR version, but the trimming step was completed.
max.multimap	numeric, default 10. If a read maps to more locations than specified, will skip the read. Set to 1 to only get unique mapping reads. Only applies for genome alignment step. The depletions are allowing for multimapping.
script.single	location of STAR single file alignment script, default internal ORFik file. You can change it and give your own if you need special alignments.

### Details

Can only run on unix systems (Linux and Mac), and requires minimum 30GB memory on genomes like human, rat, zebrafish etc.

If for some reason the internal STAR alignment bash script will not work for you, like if you have a very small genome. You can copy the internal alignment script, edit it and give that as the Index script used for this function.

The trimmer used is fastp (the fastest I could find), works on mac and linux. If you want to use your own trimmer set file1/file2 to the location of the trimmed files from your program.

A note on trimming from creator of STAR about trimming: "adapter trimming it definitely needed for short RNA sequencing. For long RNA-seq, I would agree with Devon that in most cases adapter trimming is not advantageous, since, by default, STAR performs local (not end-to-end) alignment, i.e. it auto-trims." So trimming can be skipped for longer reads.

### Value

output.dir, can be used as as input in ORFik::create.experiment

### See Also

Other STAR: [STAR.align.folder\(\)](#), [STAR.allsteps.multiQC\(\)](#), [STAR.index\(\)](#), [STAR.install\(\)](#), [STAR.multiQC\(\)](#), [STAR.remove.crashed.genome\(\)](#), [getGenomeAndAnnotation\(\)](#), [install.fastp\(\)](#)

### Examples

```
## Specify output libraries:
output.dir <- "/Bio_data/references/Human"
bam.dir <- "data/processed/human_rna_seq"
# arguments <- getGenomeAndAnnotation("Homo sapiens", output.dir)
# index <- STAR.index(arguments, output.dir)
# STAR.align.single("data/raw_data/human_rna_seq/file1.bam", bam.dir,
#                  index)
```

---

STAR.allsteps.multiQC *Create STAR multiQC plot and table*

---

### Description

Takes a folder with multiple Log.final.out files from STAR, and create a multiQC report. This is automatically run with STAR.align.folder function.

**Usage**

```
STAR.allsteps.multiQC(folder, steps = "auto")
```

**Arguments**

folder	path to main output folder of STAR run. The folder that contains /aligned/, /trim/, "contaminants_depletion" etc. To find the LOGS folders in, to use for summarized statistics.
steps	a character, default "auto". Find which steps you did. If manual, a combination of "tr-co-ge". See STAR alignment functions for description.

**Value**

data.table of main statistics, plots and data saved to disc. Named: "/00\_STAR\_LOG\_plot.png" and "/00\_STAR\_LOG\_table.csv"

**See Also**

Other STAR: [STAR.align.folder\(\)](#), [STAR.align.single\(\)](#), [STAR.index\(\)](#), [STAR.install\(\)](#), [STAR.multiQC\(\)](#), [STAR.remove.crashed.genome\(\)](#), [getGenomeAndAnnotation\(\)](#), [install.fastp\(\)](#)

---

STAR.index

*Create STAR genome index*

---

**Description**

Used as reference when aligning data  
Get genome and gtf by running `getGenomeAndFasta()`

**Usage**

```
STAR.index(
  arguments,
  output.dir = paste0(dirname(arguments[1]), "/STAR_index/"),
  star.path = STAR.install(),
  max.cpus = min(90, detectCores() - 1),
  wait = TRUE,
  remake = FALSE,
  script = system.file("STAR_Aligner", "STAR_MAKE_INDEX.sh", package = "ORFik")
)
```

**Arguments**

arguments	a named character vector containing paths wanted to use for index creation. They must be named correctly: names must be a subset of: c("gtf", "genome", "phix", "rRNA", "tRNA", "ncRNA")
output.dir	directory to save indices, default: <code>paste0(dirname(arguments[1]), "/STAR_index/")</code> , where arguments is the arguments input for this function.
star.path	path to STAR, default: <code>STAR.install()</code> , if you don't have STAR installed at default location, it will install it there, set path to a runnable star if you already have it.



<code>max.cpus</code>	integer, default: <code>min(90, detectCores() - 1)</code> , number of threads to use. Default is minimum of 90 and maximum cores - 1. So if you have 8 cores it will use 7.
<code>wait</code>	a logical (not NA) indicating whether the R interpreter should wait for the command to finish, or run it asynchronously. This will be ignored (and the interpreter will always wait) if <code>intern = TRUE</code> . When running the command asynchronously, no output will be displayed on the Rgui console in Windows (it will be dropped, instead).
<code>remake</code>	logical, default: <code>FALSE</code> , if <code>TRUE</code> remake everything specified
<code>script</code>	location of STAR index script, default internal ORFik file. You can change it and give your own if you need special alignments.

### Details

Can only run on unix systems (Linux and Mac), and requires minimum 30GB memory on genomes like human, rat, zebrafish etc.

If for some reason the internal STAR index bash script will not work for you, like if you have a very small genome. You can copy the internal index script, edit it and give that as the Index script used for this function.

### Value

`output.dir`, can be used as as input for `STAR.align..`

### See Also

Other STAR: [STAR.align.folder\(\)](#), [STAR.align.single\(\)](#), [STAR.allsteps.multiQC\(\)](#), [STAR.install\(\)](#), [STAR.multiQC\(\)](#), [STAR.remove.crashed.genome\(\)](#), [getGenomeAndAnnotation\(\)](#), [install.fastp\(\)](#)

### Examples

```
## Manual way, specify all paths yourself.
#arguments <- c(path.GTF, path.genome, path.phix, path.rrna, path.trna, path.ncrna)
#names(arguments) <- c("gtf", "genome", "phix", "rRNA", "tRNA", "ncRNA")
#STAR.index(arguments, "output.dir")

## Or use ORFik way:
output.dir <- "/Bio_data/references/Human"
# arguments <- getGenomeAndAnnotation("Homo sapiens", output.dir)
# STAR.index(arguments, output.dir)
```

---

STAR.install

*Download and prepare STAR*

---

### Description

Will not run "make", only use precompiled STAR file.

Can only run on unix systems (Linux and Mac), and requires minimum 30GB memory on genomes like human, rat, zebrafish etc.

### Usage

```
STAR.install(folder = "~/bin", version = "2.7.4a")
```

**Arguments**

folder	path to folder for download, file will be named "STAR-version", where version is version wanted.
version	default "2.7.4a"

**Value**

path to runnable STAR

**References**

<https://www.ncbi.nlm.nih.gov/pubmed/23104886>

**See Also**

Other STAR: [STAR.align.folder\(\)](#), [STAR.align.single\(\)](#), [STAR.allsteps.multiQC\(\)](#), [STAR.index\(\)](#), [STAR.multiQC\(\)](#), [STAR.remove.crashed.genome\(\)](#), [getGenomeAndAnnotation\(\)](#), [install.fastp\(\)](#)

**Examples**

```
#STAR.install("~/bin", version = "2.7.4a")
```

---

STAR.multiQC

*Create STAR multiQC plot and table*

---

**Description**

Takes a folder with multiple Log.final.out files from STAR, and create a multiQC report

**Usage**

```
STAR.multiQC(folder, type = "aligned")
```

**Arguments**

folder	path to LOGS folder of ORFik STAR runs. Can also be the path to the aligned/ (parent directory of LOGS), then it will move into LOG from there. Only if no files with pattern Log.final.out are found in parent directory. If no LOGS folder is found it can check for a folder /aligned/LOGS/ so to go 2 folders down.
type	a character path, default "aligned". Which subfolder to check for. If you want log files for contamination do type = "contaminants_depletion"

**Value**

a data.table with all information from STAR runs, plot and data saved to disc. Named: "/00\_STAR\_LOG\_plot.png" and "/00\_STAR\_LOG\_table.csv"

**See Also**

Other STAR: [STAR.align.folder\(\)](#), [STAR.align.single\(\)](#), [STAR.allsteps.multiQC\(\)](#), [STAR.index\(\)](#), [STAR.install\(\)](#), [STAR.remove.crashed.genome\(\)](#), [getGenomeAndAnnotation\(\)](#), [install.fastp\(\)](#)

---

STAR.remove.crashed.genome  
*Remove crashed STAR genome*

---

### Description

This happens if you abort STAR run early, and it halts at: ..... loading genome

### Usage

```
STAR.remove.crashed.genome(index.path, star.path = STAR.install())
```

### Arguments

index.path	path to index folder of genome
star.path	path to STAR, default: STAR.install(), if you don't have STAR installed at default location, it will install it there, set path to a runnable star if you already have it.

### Value

return value from system, 0 if all good.

### See Also

Other STAR: [STAR.align.folder\(\)](#), [STAR.align.single\(\)](#), [STAR.allsteps.multiQC\(\)](#), [STAR.index\(\)](#), [STAR.install\(\)](#), [STAR.multiQC\(\)](#), [getGenomeAndAnnotation\(\)](#), [install.fastp\(\)](#)

### Examples

```
# STAR.remove.crashed.genome(index.path = "/home/data/human_index/phix/)
```

---

startCodons	<i>Get the Start codons(3 bases) from a GRangesList of orfs grouped by orfs</i>
-------------	---

---

### Description

In ATGTTTTGA, get the positions ATG. It takes care of exons boundaries, with exons < 3 length.

### Usage

```
startCodons(grl, is.sorted = FALSE)
```

### Arguments

grl	a <a href="#">GRangesList</a> object
is.sorted	a boolean, a speedup if you know the ranges are sorted

**Value**

a GRangesList of start codons, since they might be split on exons

**See Also**

Other ORFHelpers: [defineTrailer\(\)](#), [longestORFs\(\)](#), [mapToGRanges\(\)](#), [orfID\(\)](#), [startSites\(\)](#), [stopCodons\(\)](#), [stopSites\(\)](#), [txNames\(\)](#), [uniqueGroups\(\)](#), [uniqueOrder\(\)](#)

**Examples**

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
  ranges = IRanges(c(7, 14), width = 3),
  strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
  ranges = IRanges(c(4, 1), c(9, 3)),
  strand = c("-", "-"))
grl <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
startCodons(grl, is.sorted = FALSE)
```

---

startDefinition	<i>Returns start codon definitions</i>
-----------------	--

---

**Description**

According to: <http://www.ncbi.nlm.nih.gov/Taxonomy/taxonomyhome.html/index.cgi?chapter=tgencodes#SG1> ncbi genetic code number for translation. This version is a cleaned up version, unknown indices removed.

**Usage**

```
startDefinition(transl_table)
```

**Arguments**

transl\_table    numeric. NCBI genetic code number for translation.

**Value**

A string of START sites separated with "|".

**See Also**

Other findORFs: [findMapORFs\(\)](#), [findORFsFasta\(\)](#), [findORFs\(\)](#), [findUORFs\(\)](#), [stopDefinition\(\)](#)

**Examples**

```
startDefinition
startDefinition(1)
```

---

startRegion	<i>Start region as GRangesList</i>
-------------	------------------------------------

---

### Description

Get the start region of each ORF. If you want the start codon only, set `upstream = 0` or just use [startCodons](#). Standard is 2 upstream and 2 downstream, a width 5 window centered at start site. since p-shifting is not 100 usually the reads from the start site.

### Usage

```
startRegion(grl, tx = NULL, is.sorted = TRUE, upstream = 2L, downstream = 2L)
```

### Arguments

<code>grl</code>	a <a href="#">GRangesList</a> object with usually either leaders, cds', 3' utrs or ORFs
<code>tx</code>	default NULL, a GRangesList of transcripts or (container region), names of tx must contain all grl names. The names of grl can also be the ORFik orf names. that is "txName_id"
<code>is.sorted</code>	logical (TRUE), is grl sorted.
<code>upstream</code>	an integer (2), relative region to get upstream from.
<code>downstream</code>	an integer (2), relative region to get downstream from

### Details

If tx is null, then upstream will be forced to 0 and downstream to a maximum of grl width (3' UTR end for mRNAs). Since there is no reference for splicing.

### Value

a GRanges, or GRangesList object if any group had > 1 exon.

### See Also

Other features: [computeFeaturesCage\(\)](#), [computeFeatures\(\)](#), [countOverlapsW\(\)](#), [disengagementScore\(\)](#), [distToCds\(\)](#), [distToTSS\(\)](#), [entropy\(\)](#), [floss\(\)](#), [fpkm\\_calc\(\)](#), [fpkm\(\)](#), [fractionLength\(\)](#), [initiationScore\(\)](#), [insideOutsideORF\(\)](#), [isInFrame\(\)](#), [isOverlapping\(\)](#), [kozakSequenceScore\(\)](#), [orfScore\(\)](#), [rankOrder\(\)](#), [ribosomeReleaseScore\(\)](#), [ribosomeStallingScore\(\)](#), [startRegionCoverage\(\)](#), [stopRegion\(\)](#), [subsetCoverage\(\)](#), [translationalEff\(\)](#)

---

startRegionCoverage    *Start region coverage*

---

### Description

Get the number of reads in the start region of each ORF. If you want the start codon coverage only, set upstream = 0. Standard is 2 upstream and 2 downstream, a width 5 window centered at start site. since p-shifting is not 100 start site.

### Usage

```
startRegionCoverage(
  grl,
  RFP,
  tx = NULL,
  is.sorted = TRUE,
  upstream = 2L,
  downstream = 2L,
  weight = 1L
)
```

### Arguments

grl	a <a href="#">GRangesList</a> object with usually either leaders, cds', 3' utrs or ORFs
RFP	ribo seq reads as <a href="#">GAlignments</a> , <a href="#">GRanges</a> or <a href="#">GRangesList</a> object
tx	default NULL, a <a href="#">GRangesList</a> of transcripts or (container region), names of tx must contain all grl names. The names of grl can also be the ORFik orf names. that is "txName_id"
is.sorted	logical (TRUE), is grl sorted.
upstream	an integer (2), relative region to get upstream from.
downstream	an integer (2), relative region to get downstream from
weight	a vector (default: 1L, if 1L it is identical to <code>countOverlaps()</code> ), if single number ( $\neq 1$ ), it applies for all, if more than one must be equal size of 'reads'. else it must be the string name of a defined meta column in subject "reads", that gives number of times a read was found. <code>GRanges("chr1", 1, "+", score = 5)</code> , would mean "score" column tells that this alignment region was found 5 times.

### Details

If tx is null, then upstream will be force to 0 and downstream to a maximum of grl width. Since there is no reference for splicing.

### Value

a numeric vector of counts

**See Also**

Other features: [computeFeaturesCage\(\)](#), [computeFeatures\(\)](#), [countOverlapsW\(\)](#), [disengagementScore\(\)](#), [distToCds\(\)](#), [distToTSS\(\)](#), [entropy\(\)](#), [floss\(\)](#), [fpkm\\_calc\(\)](#), [fpkm\(\)](#), [fractionLength\(\)](#), [initiationScore\(\)](#), [insideOutsideORF\(\)](#), [isInFrame\(\)](#), [isOverlapping\(\)](#), [kozakSequenceScore\(\)](#), [orfScore\(\)](#), [rankOrder\(\)](#), [ribosomeReleaseScore\(\)](#), [ribosomeStallingScore\(\)](#), [startRegion\(\)](#), [stopRegion\(\)](#), [subsetCoverage\(\)](#), [translationalEff\(\)](#)

---

startRegionString	<i>Get start region as DNA-strings per GRanges group</i>
-------------------	--

---

**Description**

One window per start site, if upstream and downstream are both 0, then only the startsite is returned.

**Usage**

```
startRegionString(grl, tx, faFile, upstream = 20, downstream = 20)
```

**Arguments**

grl	a <a href="#">GRangesList</a> of ranges to find regions in.
tx	a <a href="#">GRangesList</a> of transcripts or (container region), names of tx must contain all gr names. The names of gr can also be the ORFik orf names. that is "tx-Name_id".
faFile	<a href="#">FaFile</a> , BSgenome, fasta/index file path or an ORFik <a href="#">experiment</a> . This file is usually used to find the transcript sequences from some <a href="#">GRangesList</a> .
upstream	an integer, default (0), relative region to get upstream from.
downstream	an integer, default (0), relative region to get downstream from

**Value**

a character vector of start regions

---

startSites	<i>Get the start sites from a GRangesList of orfs grouped by orfs</i>
------------	---

---

**Description**

In ATGTTTTGG, get the position of the A.

**Usage**

```
startSites(grl, asGR = FALSE, keep.names = FALSE, is.sorted = FALSE)
```

**Arguments**

gr1	a <a href="#">GRangesList</a> object
asGR	a boolean, return as GRanges object
keep.names	a logical (FALSE), keep names of input.
is.sorted	a speedup, if you know the ranges are sorted

**Value**

if asGR is False, a vector, if True a GRanges object

**See Also**

Other ORFHelpers: [defineTrailer\(\)](#), [longestORFs\(\)](#), [mapToGRanges\(\)](#), [orfID\(\)](#), [startCodons\(\)](#), [stopCodons\(\)](#), [stopSites\(\)](#), [txNames\(\)](#), [uniqueGroups\(\)](#), [uniqueOrder\(\)](#)

**Examples**

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
  ranges = IRanges(c(7, 14), width = 3),
  strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
  ranges = IRanges(c(4, 1), c(9, 3)),
  strand = c("-", "-"))
gr1 <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
startSites(gr1, is.sorted = FALSE)
```

---

stopCodons	<i>Get the Stop codons (3 bases) from a GRangesList of orfs grouped by orfs</i>
------------	---

---

**Description**

In ATGTTTTGA, get the positions TGA. It takes care of exons boundaries, with exons < 3 length.

**Usage**

```
stopCodons(gr1, is.sorted = FALSE)
```

**Arguments**

gr1	a <a href="#">GRangesList</a> object
is.sorted	a boolean, a speedup if you know the ranges are sorted

**Value**

a GRangesList of stop codons, since they might be split on exons

**See Also**

Other ORFHelpers: [defineTrailer\(\)](#), [longestORFs\(\)](#), [mapToGRanges\(\)](#), [orfID\(\)](#), [startCodons\(\)](#), [startSites\(\)](#), [stopSites\(\)](#), [txNames\(\)](#), [uniqueGroups\(\)](#), [uniqueOrder\(\)](#)



## Examples

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
  ranges = IRanges(c(7, 14), width = 3),
  strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
  ranges = IRanges(c(4, 1), c(9, 3)),
  strand = c("-", "-"))
grl <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
stopCodons(grl, is.sorted = FALSE)
```

---

stopDefinition	Returns stop codon definitions
----------------	--------------------------------

---

## Description

According to: <<http://www.ncbi.nlm.nih.gov/Taxonomy/taxonomyhome.html/index.cgi?chapter=tgencodes#SG1>> ncbi genetic code number for translation. This version is a cleaned up version, unknown indices removed.

## Usage

```
stopDefinition(transl_table)
```

## Arguments

transl\_table    numeric. NCBI genetic code number for translation.

## Value

A string of STOP sites separated with "|".

## See Also

Other findORFs: [findMapORFs\(\)](#), [findORFsFasta\(\)](#), [findORFs\(\)](#), [findUORFs\(\)](#), [startDefinition\(\)](#)

## Examples

```
stopDefinition
stopDefinition(1)
```

---

stopRegion	<i>Stop region as GRangesList</i>
------------	-----------------------------------

---

### Description

Get the stop region of each ORF / region. If you want the stop codon only, set upstream = 0 or just use [stopCodons](#). Standard is 2 upstream and 2 downstream, a width 5 window centered at stop site.

### Usage

```
stopRegion(grl, tx = NULL, is.sorted = TRUE, upstream = 2L, downstream = 2L)
```

### Arguments

grl	a <a href="#">GRangesList</a> object with usually either leaders, cds', 3' utrs or ORFs
tx	default NULL, a GRangesList of transcripts or (container region), names of tx must contain all grl names. The names of grl can also be the ORFik orf names. that is "txName_id"
is.sorted	logical (TRUE), is grl sorted.
upstream	an integer (2), relative region to get upstream from.
downstream	an integer (2), relative region to get downstream from

### Details

If tx is null, then downstream will be forced to 0 and upstream to a minimum of -grl width (to the TSS). . Since there is no reference for splicing.

### Value

a GRanges, or GRangesList object if any group had > 1 exon.

### See Also

Other features: [computeFeaturesCage\(\)](#), [computeFeatures\(\)](#), [countOverlapsW\(\)](#), [disengagementScore\(\)](#), [distToCds\(\)](#), [distToTSS\(\)](#), [entropy\(\)](#), [floss\(\)](#), [fpkm\\_calc\(\)](#), [fpkm\(\)](#), [fractionLength\(\)](#), [initiationScore\(\)](#), [insideOutsideORF\(\)](#), [isInFrame\(\)](#), [isOverlapping\(\)](#), [kozakSequenceScore\(\)](#), [orfScore\(\)](#), [rankOrder\(\)](#), [ribosomeReleaseScore\(\)](#), [ribosomeStallingScore\(\)](#), [startRegionCoverage\(\)](#), [startRegion\(\)](#), [subsetCoverage\(\)](#), [translationalEff\(\)](#)

---

stopSites	<i>Get the stop sites from a GRangesList of orfs grouped by orfs</i>
-----------	--

---

**Description**

In ATGTTTTGC, get the position of the C.

**Usage**

```
stopSites(grl, asGR = FALSE, keep.names = FALSE, is.sorted = FALSE)
```

**Arguments**

grl	a <a href="#">GRangesList</a> object
asGR	a boolean, return as GRanges object
keep.names	a logical (FALSE), keep names of input.
is.sorted	a speedup, if you know the ranges are sorted

**Value**

if asGR is False, a vector, if True a GRanges object

**See Also**

Other ORFHelpers: [defineTrailer\(\)](#), [longestORFs\(\)](#), [mapToGRanges\(\)](#), [orfID\(\)](#), [startCodons\(\)](#), [startSites\(\)](#), [stopCodons\(\)](#), [txNames\(\)](#), [uniqueGroups\(\)](#), [uniqueOrder\(\)](#)

**Examples**

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
  ranges = IRanges(c(7, 14), width = 3),
  strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
  ranges = IRanges(c(4, 1), c(9, 3)),
  strand = c("-", "-"))
grl <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
stopSites(grl, is.sorted = FALSE)
```

---

strandBool	<i>Get logical list of strands</i>
------------	------------------------------------

---

**Description**

Helper function to get a logical list of True/False, if GRangesList group have + strand = T, if - strand = F Also checks for \* strands, so a good check for bugs

**Usage**

```
strandBool(grl)
```

**Arguments**

gr1                    a [GRangesList](#) or GRanges object

**Value**

a logical vector

**Examples**

```
gr <- GRanges(Rle(c("chr2", "chr2", "chr1", "chr3"), c(1, 3, 2, 4)),
              IRanges(1:10, width = 10:1),
              Rle(strand(c("-", "+", "*", "+", "-")), c(1, 2, 2, 3, 2)))
strandBool(gr)
```

---

strandPerGroup	<i>Get list of strands per granges group</i>
----------------	--

---

**Description**

Get list of strands per granges group

**Usage**

```
strandPerGroup(gr1, keep.names = TRUE)
```

**Arguments**

gr1                    a [GRangesList](#)  
 keep.names            a boolean, keep names or not, default: (TRUE)

**Value**

a vector named/unnamed of characters

**Examples**

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
                  ranges = IRanges(c(7, 14), width = 3),
                  strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
                   ranges = IRanges(c(4, 1), c(9, 3)),
                   strand = c("-", "-"))
gr1 <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
strandPerGroup(gr1)
```

---

subsetCoverage                      *Subset GRanges to get coverage.*

---

### Description

GRanges object should be beforehand tiled to size of 1. This subsetting takes account for strand.

### Usage

```
subsetCoverage(cov, y)
```

### Arguments

cov	A coverage object from coverage()
y	GRanges object for which coverage should be extracted

### Value

numeric vector of coverage of input GRanges object

### See Also

Other features: [computeFeaturesCage\(\)](#), [computeFeatures\(\)](#), [countOverlapsW\(\)](#), [disengagementScore\(\)](#), [distToCds\(\)](#), [distToTSS\(\)](#), [entropy\(\)](#), [floss\(\)](#), [fpkm\\_calc\(\)](#), [fpkm\(\)](#), [fractionLength\(\)](#), [initiationScore\(\)](#), [insideOutsideORF\(\)](#), [isInFrame\(\)](#), [isOverlapping\(\)](#), [kozakSequenceScore\(\)](#), [orfScore\(\)](#), [rankOrder\(\)](#), [ribosomeReleaseScore\(\)](#), [ribosomeStallingScore\(\)](#), [startRegionCoverage\(\)](#), [startRegion\(\)](#), [stopRegion\(\)](#), [translationalEff\(\)](#)

---

subsetToFrame                      *Subset GRanges to get desired frame.*

---

### Description

Usually used for ORFs to get specific frame (0-2): frame 0, frame 1, frame 2

### Usage

```
subsetToFrame(x, frame)
```

### Arguments

x	A tiled to size of 1 GRanges object
frame	A numeric indicating which frame to extract

### Details

GRanges object should be beforehand tiled to size of 1. This subsetting takes account for strand.

### Value

GRanges object reduced to only first frame

---

te.plot *Translational efficiency plots*

---

### Description

Create 2 TE plots of:

- Within sample (TE log2 vs mRNA fpkm) ("default")
- Between all combinations of samples (x-axis: rna1fpkm - rna2fpkm, y-axis rfp1fpkm - rfp2fpkm)

### Usage

```
te.plot(
  df.rfp,
  df.rna,
  output.dir = paste0(dirname(df.rfp$filepath[1]), "/QC_STATS/"),
  type = c("default", "between"),
  filter.rfp = 1,
  filter.rna = 1,
  collapse = FALSE,
  plot.title = "",
  width = 6,
  height = "auto"
)
```

### Arguments

df.rfp	a <a href="#">experiment</a> of Ribo-seq or 80S from TCP-seq.
df.rna	a <a href="#">experiment</a> of RNA-seq
output.dir	directory to save plots, plots will be named "TE_between.png" and "TE_within.png"
type	which plots to make, default: c("default", "between"). Both plots.
filter.rfp	numeric, default 1. minimum fpkm value to be included in plots
filter.rna	numeric, default 1. minimum fpkm value to be included in plots
collapse	a logical/character (default FALSE), if TRUE all samples within the group SAMPLE will be collapsed to one. If "all", all groups will be merged into 1 column called merged_all. Collapse is defined as $\text{rowSum}(\text{elements\_per\_group}) / \text{ncol}(\text{elements\_per\_group})$
plot.title	title for plots, usually name of experiment etc
width	numeric, default 6 (in inches)
height	numeric or character, default "auto", which is: $3 + (\text{ncol}(\text{RFP\_CDS\_FPKM}) - 2)$ . Else a numeric value of height (in inches)

### Details

Ribo-seq and RNA-seq must have equal nrows, with matching samples. Only exception is if RNA-seq is 1 single sample. Then it will use that for each of the Ribo-seq samples. Same stages, conditions etc, with a unique pairing 1 to 1. If not you can run collapse = "all". It will then merge all and do combined of all RNA-seq vs all Ribo-seq

**Value**

a data.table with TE values, fpkm and log fpkm values, library samples melted into rows with split variable called "variable".

**Examples**

```
##
# df.rfp <- read.experiment("zf_baz14_RFP")
# df.rna <- read.experiment("zf_baz14_RNA")
# te.plot(df.rfp, df.rna)
## Collapse replicates:
# te.plot(df.rfp, df.rna, collapse = TRUE)
```

---

te.table	<i>Create a TE table</i>
----------	--------------------------

---

**Description**

Creates a data.table with 5 columns, column names are:  
variable, rfp\_log2, rna\_log2, rna\_log10, LFC\_TE

**Usage**

```
te.table(df.rfp, df.rna, filter.rfp = 1, filter.rna = 1, collapse = FALSE)
```

**Arguments**

df.rfp	a <a href="#">experiment</a> of Ribo-seq or 80S from TCP-seq.
df.rna	a <a href="#">experiment</a> of RNA-seq
filter.rfp	numeric, default 1. What is the minimum fpkm value?
filter.rna	numeric, default 1. What is the minimum fpkm value?
collapse	a logical/character (default FALSE), if TRUE all samples within the group SAMPLE will be collapsed to one. If "all", all groups will be merged into 1 column called merged_all. Collapse is defined as $\text{rowSum}(\text{elements\_per\_group}) / \text{ncol}(\text{elements\_per\_group})$

**Value**

a data.table with 5 columns

**See Also**

Other TE: [DTEG.analysis\(\)](#), [DTEG.plot\(\)](#), [te\\_rna.plot\(\)](#)

---

te_rna.plot	<i>Translational efficiency plots</i>
-------------	---------------------------------------

---

### Description

Create TE plot of:  
 - Within sample (TE log2 vs mRNA fpkm)

### Usage

```
te_rna.plot(
  dt,
  output.dir = NULL,
  filter.rfp = 1,
  filter.rna = 1,
  plot.title = "",
  width = 6,
  height = "auto",
  dot.size = 0.4
)
```

### Arguments

dt	a data.table with the results from <a href="#">te.table</a>
output.dir	a character path, default NULL(no save), or a directory to save to a file will be called "TE_within.png"
filter.rfp	numeric, default 1. What is the minimum fpkm value?
filter.rna	numeric, default 1. What is the minimum fpkm value?
plot.title	title for plots, usually name of experiment etc
width	numeric, default 6 (in inches)
height	a numeric, width of plot in inches. Default "auto".
dot.size	numeric, default 0.4, size of point dots in plot.

### Value

a ggplot object

### See Also

Other TE: [DTEG.analysis\(\)](#), [DTEG.plot\(\)](#), [te.table\(\)](#)

### Examples

```
#df.rfp <- read.experiment("Riboseq")
#df.rna <- read.experiment("RNAseq")
#dt <- te.table(df.rfp, df.rna)
#te_rna.plot(dt)
```



---

tile1	<i>Tile each GRangesList group to 1-base resolution.</i>
-------	--

---

### Description

Will tile a GRangesList into single bp resolution, each group of the list will be splited by positions of 1. Returned values are sorted as the same groups as the original GRangesList, except they are in bp resolutions. This is not supported originally by GenomicRanges for GRangesList.

### Usage

```
tile1(gr1, sort.on.return = TRUE, matchNaming = TRUE)
```

### Arguments

`gr1` a [GRangesList](#) object with names

`sort.on.return` logical (T), should the groups be sorted before return.

`matchNaming` logical (T), should groups keep unlisted names and meta data.(This make the list very big, for > 100K groups)

### Value

a GRangesList grouped by original group, tiled to 1

### See Also

Other ExtendGenomicRanges: [asTX\(\)](#), [coveragePerTiling\(\)](#), [extendLeaders\(\)](#), [extendTrailers\(\)](#), [reduceKeepAttr\(\)](#), [txSeqsFromFa\(\)](#), [windowPerGroup\(\)](#)

### Examples

```
gr1 <- GRanges("1", ranges = IRanges(start = c(1, 10, 20),
                                     end = c(5, 15, 25)),
              strand = "+")
gr2 <- GRanges("1", ranges = IRanges(start = c(20, 30, 40),
                                     end = c(25, 35, 45)),
              strand = "+")
names(gr1) = rep("tx1_1", 3)
names(gr2) = rep("tx1_2", 3)
gr1 <- GRangesList(tx1_1 = gr1, tx1_2 = gr2)
tile1(gr1)
```

---

tissueNames	<i>Get tissue name variants</i>
-------------	---------------------------------

---

**Description**

Used to standardize nomenclature for experiments.

Example: testis is main naming, but a variant is testicles. testicles will then be renamed to testis.

**Usage**

```
tissueNames()
```

**Value**

a data.table with 2 columns, the main name, and all name variants of the main name in second column as a list.

**See Also**

Other experiment\_naming: [cellLineNames\(\)](#), [conditionNames\(\)](#), [libNames\(\)](#), [mainNames\(\)](#), [repNames\(\)](#), [stageNames\(\)](#)

---

TOP.Motif.ecdf	<i>TOP Motif ecdf plot</i>
----------------	----------------------------

---

**Description**

Given sequences, DNA or RNA. And some score, scanning efficiency (SE), ribo-seq fpkm, TE etc.

**Usage**

```
TOP.Motif.ecdf(  
  seqs,  
  rate,  
  start = 1,  
  stop = max(nchar(seqs)),  
  xlim = c("q10", "q99"),  
  type = "Scanning efficiency",  
  legend.position.1st = c(0.75, 0.28),  
  legend.position.motif = c(0.75, 0.28)  
)
```

**Arguments**

seqs	the sequences (character vector, DNAStrngSet), of 5' UTRs (leaders). See example below for input.
rate	a scoring vector (equal size to seqs)
start	position in seqs to start at (first is 1), default 1.

stop	position in seqs to stop at (first is 1), default max(nchar(seqs)), that is the longest sequence length
xlim	What interval of rate values you want to show type: numeric or quantile of length 2, 1. default c("q10","q99"). bigger than 10 percentile and less than 99 percentile. 2. Set to numeric values, like c(5, 1000), 3. Set to NULL if you want all values. Backend uses coord_cartesian.
type	What type is the rate scoring ? default ("Scanning efficiency")
legend.position.1st	adjust left plot label position, default c(0.75, 0.28), ("none", "left", "right", "bottom", "top", or two-element numeric vector)
legend.position.motif	adjust right plot label position, default c(0.75, 0.28), ("none", "left", "right", "bottom", "top", or two-element numeric vector)

### Details

Top motif defined as a TSS of C and 4 T's or C's (pyrimidins) downstream of TSS C.

The right plot groups: C nucleotide, TOP motif (C, then 4 pyrimidines) and OTHER (all other TSS variants).

### Value

a ggplot gtable of the TOP motifs in 2 plots

### Examples

```
## Not run:
if (requireNamespace("BSgenome.Hsapiens.UCSC.hg19")) {
  txdbFile <- system.file("extdata", "hg19_knownGene_sample.sqlite",
    package = "GenomicFeatures")
  #Extract sequences of Coding sequences.
  leaders <- loadRegion(txdbFile, "leaders")

  # Should update by CAGE if not already done
  cageData <- system.file("extdata", "cage-seq-heart.bed.bgz",
    package = "ORFik")
  leadersCage <- reassignTSSbyCage(leaders, cageData)
  # Get region to check
  seqs <- startRegionString(leadersCage, NULL,
    BSgenome.Hsapiens.UCSC.hg19::Hsapiens, 0, 4)
  # Some toy ribo-seq fpkm scores on cds
  set.seed(3)
  fpkm <- sample(1:115, length(leadersCage), replace = TRUE)
  # Standard arguments
  TOP.Motif.ecdf(seqs, fpkm, type = "ribo-seq FPKM",
    legend.position.1st = "bottom",
    legend.position.motif = "bottom")
  # with no zoom on x-axis:
  TOP.Motif.ecdf(seqs, fpkm, xlim = NULL,
    legend.position.1st = "bottom",
    legend.position.motif = "bottom")
}
```

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

---

```
topMotif          TOP Motif detection
```

---

### Description

Per leader, detect if the leader has a TOP motif at TSS (5' end of leader) TOP motif defined as: (C, then 4 pyrimidines)

### Usage

```
topMotif(seqs, start = 1, stop = max(nchar(seqs)), return.sequence = TRUE)
```

### Arguments

seqs	the sequences (character vector, DNASTringSet), of 5' UTRs (leaders) start region. seqs must be of minimum widths start - stop + 1 to be included. See example below for input.
start	position in seqs to start at (first is 1), default 1.
stop	position in seqs to stop at (first is 1), default max(nchar(seqs)), that is the longest sequence length
return.sequence	logical, default TRUE, return as data.table with sequence as columns in addition to TOP class. If FALSE, return character vector.

### Value

default: return.sequence == FALSE, a character vector of either TOP, C or OTHER. C means leaders started on C, Other means not TOP and did not start on C. If return.sequence == TRUE, a data.table is returned with the base per position in the motif is included as additional columns (per position called seq1, seq2 etc) and a id column called X.gene\_id (with names of seqs).

### Examples

```
## Not run:
if (requireNamespace("BSgenome.Hsapiens.UCSC.hg19")) {
  txdbFile <- system.file("extdata", "hg19_knownGene_sample.sqlite",
    package = "GenomicFeatures")
  #Extract sequences of Coding sequences.
  leaders <- loadRegion(txdbFile, "leaders")

  # Should update by CAGE if not already done
  cageData <- system.file("extdata", "cage-seq-heart.bed.bgz",
    package = "ORFik")
  leadersCage <- reassignTSSbyCage(leaders, cageData)
  # Get region to check
  seqs <- startRegionString(leadersCage, NULL,
    BSgenome.Hsapiens.UCSC.hg19::Hsapiens, 0, 4)
  topMotif(seqs)
```

```

}
## End(Not run)

```

---

transcriptWindow	<i>Make 100 bases size meta window for all libraries in experiment</i>
------------------	--

---

## Description

Gives you binned meta coverage plots, either saved separately or all in one.

## Usage

```

transcriptWindow(
  leaders,
  cds,
  trailers,
  df,
  outdir = NULL,
  scores = c("sum", "zscore"),
  allTogether = TRUE,
  colors = experiment.colors(df),
  title = "Coverage metaplot",
  windowSize = min(100, min(widthPerGroup(leaders, FALSE)), min(widthPerGroup(cds,
    FALSE)), min(widthPerGroup(trailers, FALSE))),
  returnPlot = is.null(outdir),
  dfr = NULL,
  idName = "",
  format = ".png",
  type = "ofst",
  BPPARAM = bpparam()
)

```

## Arguments

leaders	a <a href="#">GRangesList</a> of leaders (5' UTRs)
cds	a <a href="#">GRangesList</a> of coding sequences
trailers	a <a href="#">GRangesList</a> of trailers (3' UTRs)
df	an <a href="#">ORFik</a> <a href="#">experiment</a>
outdir	directory to save to (default: NULL, no saving)
scores	scoring function (default: c("sum", "zscore")), see <a href="#">?coverageScorings</a> for possible scores.
allTogether	plot all coverage plots in 1 output? (default: TRUE)
colors	Which colors to use, default auto color from function <a href="#">experiment.colors</a> , new color per library type. Else assign colors yourself.
title	title of ggplot
windowSize	size of binned windows, default: 100
returnPlot	return plot from function, default is.null(outdir), so TRUE if outdir is not defined.

dfr	an ORFik <a href="#">experiment</a> of RNA-seq to normalize against. Will add RNA normalized to plot name if this is done.
idName	A character ID to add to saved name of plot, if you make several plots in the same folder, and same experiment, like splitting transcripts in two groups like targets / nontargets etc. (default: "")
format	default (".png"), do ".pdf" if you want as pdf
type	a character(default: "bedoc"), load files in experiment or some precomputed variant, either "bedo", "bedoc", "pshifted" or default. These are made with ORFik:::simpleLibs(), shiftFootprintsByExperiment().. Will load default if bedoc is not found
BPPARAM	how many cores/threads to use? default: bpparam()

**Value**

NULL, or ggplot object if returnPlot is TRUE

**See Also**

Other experiment plots: [transcriptWindow1\(\)](#), [transcriptWindowPer\(\)](#)

**Examples**

```
df <- ORFik.template.experiment()[3,] # Only third library
loadRegions(df) # Load leader, cds and trailers as GRangesList
#transcriptWindow(leaders, cds, trailers, df, outdir = "directory/to/save")
```

---

transcriptWindow1      *Meta coverage over all transcripts*

---

**Description**

Given as single window

**Usage**

```
transcriptWindow1(
  df,
  outdir = NULL,
  scores = c("sum", "zscore"),
  colors = experiment.colors(df),
  title = "Coverage metaplot",
  windowSize = 100,
  returnPlot = is.null(outdir),
  dfr = NULL,
  idName = "",
  format = ".png",
  type = "ofst",
  BPPARAM = bpparam()
)
```

**Arguments**

df	an ORFik <a href="#">experiment</a>
outdir	directory to save to (default: NULL, no saving)
scores	scoring function (default: c("sum", "zscore")), see <a href="#">?coverageScorings</a> for possible scores.
colors	Which colors to use, default auto color from function <a href="#">experiment.colors</a> , new color per library type. Else assign colors yourself.
title	title of ggplot
windowSize	size of binned windows, default: 100
returnPlot	return plot from function, default is.null(outdir), so TRUE if outdir is not defined.
dfr	an ORFik <a href="#">experiment</a> of RNA-seq to normalize against. Will add RNA normalized to plot name if this is done.
idName	A character ID to add to saved name of plot, if you make several plots in the same folder, and same experiment, like splitting transcripts in two groups like targets / nontargets etc. (default: "")
format	default (".png"), do ".pdf" if you want as pdf
type	a character(default: "bedoc"), load files in experiment or some precomputed variant, either "bedo", "bedoc", "pshifted" or default. These are made with ORFik:::simpleLibs(), shiftFootprintsByExperiment().. Will load default if bedoc is not found
BPPARAM	how many cores/threads to use? default: bpparam()

**Value**

NULL, or ggplot object if returnPlot is TRUE

**See Also**

Other experiment plots: [transcriptWindowPer\(\)](#), [transcriptWindow\(\)](#)

---

transcriptWindowPer     *Helper function for transcriptWindow*

---

**Description**

Make 100 bases size meta window for one library in experiment

**Usage**

```
transcriptWindowPer(
  leaders,
  cds,
  trailers,
  df,
  outdir = NULL,
  scores = c("sum", "zscore"),
```

```

reads,
returnCoverage = FALSE,
windowSize = 100,
BPPARAM = bpparam()
)

```

### Arguments

leaders	a <a href="#">GRangesList</a> of leaders (5' UTRs)
cds	a <a href="#">GRangesList</a> of coding sequences
trailers	a <a href="#">GRangesList</a> of trailers (3' UTRs)
df	an <a href="#">ORFik experiment</a>
outdir	directory to save to (default: NULL, no saving)
scores	scoring function (default: c("sum", "zscore")), see <a href="#">?coverageScorings</a> for possible scores.
reads	a <a href="#">GRanges</a> / <a href="#">GAlignment</a> object of reads, can also be a list of those.
returnCoverage	return data.table with coverage (default: FALSE)
windowSize	size of binned windows, default: 100
BPPARAM	how many cores/threads to use? default: <a href="#">bpparam()</a>

### Details

Gives you binned meta coverage plots, either saved separately or all in one.

### Value

NULL, or ggplot object if `returnPlot` is TRUE

### See Also

Other experiment plots: [transcriptWindow1\(\)](#), [transcriptWindow\(\)](#)

---

translationalEff	<i>Translational efficiency</i>
------------------	---------------------------------

---

### Description

Uses RnaSeq and RiboSeq to get translational efficiency of every element in 'grl'. Translational efficiency is defined as:

$$(\text{density of RPF within ORF}) / (\text{RNA expression of ORFs transcript})$$



**Usage**

```
translationalEff(
  grl,
  RNA,
  RFP,
  tx,
  with.fpkm = FALSE,
  pseudoCount = 0,
  librarySize = "full",
  weight.RFP = 1L,
  weight.RNA = 1L
)
```

**Arguments**

grl	a <a href="#">GRangesList</a> object can be either transcripts, 5' utrs, cds', 3' utrs or ORFs as a special case (uORFs, potential new cds' etc). If regions are not spliced you can send a <a href="#">GRanges</a> object.
RNA	RnaSeq reads as <a href="#">GAlignments</a> , <a href="#">GRanges</a> or <a href="#">GRangesList</a> object
RFP	RiboSeq reads as <a href="#">GAlignments</a> , <a href="#">GRanges</a> or <a href="#">GRangesList</a> object
tx	a <a href="#">GRangesList</a> of the transcripts. If you used cage data, then the tss for the leaders have changed, therefor the tx lengths have changed. To account for that call: ' translationalEff(grl, RNA, RFP, tx = extendLeaders(tx, cageFiveUTRs)) ' where cageFiveUTRs are the reannotated by CageSeq data leaders.
with.fpkm	logical, default: FALSE, if true return the fpkm values together with translational efficiency as a data.table
pseudoCount	an integer, by default is 0, set it to 1 if you want to avoid NA and inf values.
librarySize	either numeric value or character vector. Default ("full"), number of alignments in library (reads). If you just have a subset, you can give the value by librarySize = length(wholeLib), if you want lib size to be only number of reads overlapping grl, do: librarySize = "overlapping" sum(countOverlaps(reads, grl) > 0), if reads[1] has 3 hits in grl, and reads[2] has 2 hits, librarySize will be 2, not 5. You can also get the inverse overlap, if you want lib size to be total number of overlaps, do: librarySize = "DESeq" This is standard fpkm way of DESeq2::fpkm(robust = FALSE) sum(countOverlaps(grl, reads)) if grl[1] has 3 reads and grl[2] has 2 reads, librarySize is 5, not 2.
weight.RFP	a vector (default: 1L). Can also be character name of column in RFP. As in translationalEff(weight = "score") for: GRanges("chr1", 1, "+", score = 5), would mean score column tells that this alignment region was found 5 times.
weight.RNA	Same as weightRFP but for RNA weights. (default: 1L)

**Value**

a numeric vector of fpkm ratios, if with.fpkm is TRUE, return a data.table with te and fpkm values (total 3 columns then)

**References**

doi: 10.1126/science.1168978

**See Also**

Other features: `computeFeaturesCage()`, `computeFeatures()`, `countOverlapsW()`, `disengagementScore()`, `distToCds()`, `distToTSS()`, `entropy()`, `floss()`, `fpkm_calc()`, `fpkm()`, `fractionLength()`, `initiationScore()`, `insideOutsideORF()`, `isInFrame()`, `isOverlapping()`, `kozakSequenceScore()`, `orfScore()`, `rankOrder()`, `ribosomeReleaseScore()`, `ribosomeStallingScore()`, `startRegionCoverage()`, `startRegion()`, `stopRegion()`, `subsetCoverage()`

**Examples**

```
ORF <- GRanges(seqnames = "1",
               ranges = IRanges(start = c(1, 10, 20), end = c(5, 15, 25)),
               strand = "+")
grl <- GRangesList(tx1_1 = ORF)
RFP <- GRanges("1", IRanges(25, 25), "+")
RNA <- GRanges("1", IRanges(1, 50), "+")
tx <- GRangesList(tx1 = GRanges("1", IRanges(1, 50), "+"))
# grl must have same names as cds + _1 etc, so that they can be matched.
te <- translationalEff(grl, RNA, RFP, tx, with.fpkm = TRUE, pseudoCount = 1)
te$fpkmRFP
te$te
```

---

trimming.table	<i>Create trimming table</i>
----------------	------------------------------

---

**Description**

From fastp runs in ORFik alignment process

**Usage**

```
trimming.table(trim_folder)
```

**Arguments**

`trim_folder`      folder of trimmed files

**Value**

a data.table with 4 columns, `raw_library` (names of library), `raw_reads` (numeric, number of raw reads), `trim_reads` (numeric, number of trimmed reads),

---

trim_detection	<i>Add trimming info to QC report</i>
----------------	---------------------------------------

---

**Description**

Only works if alignment was done using ORFik with STAR.

**Usage**

```
trim_detection(df, finals, out.dir)
```

**Arguments**

df	an ORFik <a href="#">experiment</a>
finals	a data.table with current output from QCreport
out.dir	optional output directory, default: <code>dirname(df\$filepath[1])</code> . Will make a folder called "QC_STATS" with all results in this directory.

**Value**

a data.table of the update finals object with trim info

---

txNames	<i>Get transcript names from orf names</i>
---------	--

---

**Description**

Using the ORFik definition of orf name, which is: example ENSEMBL: tx name: ENST09090909090 orf id: \_1 (the first of on that tx) orf\_name: ENST09090909090\_1 So therefor `txNames("ENST09090909090_1") = ENST09090909090`

**Usage**

```
txNames(grl, ref = NULL, unique = FALSE)
```

**Arguments**

grl	a <a href="#">GRangesList</a> grouped by ORF , GRanges object or IRanges object.
ref	a reference <a href="#">GRangesList</a> . The object you want grl to subset by names. Add to make sure naming is valid.
unique	a boolean, if true unique the names, used if several orfs map to same transcript and you only want the unique groups

**Details**

The names must be extracted from a column called names, or the names of the grl object. If it is already tx names, it returns the input

NOTE! Do not use \_123 etc in end of transcript names if it is not ORFs. Else you will get errors. Just \_ will work, but if transcripts are called ENST\_123124124000 etc, it will crash, so substitute "\_" with "." `gsub("_", ".", names)`

**Value**

a character vector of transcript names, without `_*` naming

**See Also**

Other ORFHelpers: `defineTrailer()`, `longestORFs()`, `mapToGRanges()`, `orfID()`, `startCodons()`, `startSites()`, `stopCodons()`, `stopSites()`, `uniqueGroups()`, `uniqueOrder()`

**Examples**

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
                  ranges = IRanges(c(7, 14), width = 3),
                  strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
                   ranges = IRanges(c(4, 1), c(9, 3)),
                   strand = c("-", "-"))
grl <- GRangesList(tx1_1 = gr_plus, tx2_1 = gr_minus)
# there are 2 orfs, both the first on each transcript
txNames(grl)
```

---

txNamesToGeneNames	<i>Convert transcript names to gene names</i>
--------------------	---

---

**Description**

Works for ensembl, UCSC and other standard annotations.

**Usage**

```
txNamesToGeneNames(txNames, txdb)
```

**Arguments**

txNames	character vector, the transcript names to convert. Can also be a named object with tx names (like a GRangesList), will then extract names.
txdb	the transcript database to use or gtf/gff path to it.

**Value**

character vector of gene names

**Examples**

```
gtf <- system.file("extdata", "annotations.gtf", package = "ORFik")
txdb <- loadTxdb(gtf)
loadRegions(txdb, "cds") # using tx names
txNamesToGeneNames(cds, txdb)
# Identical to:
loadRegions(txdb, "cds", by = "gene")
```

---

txSeqsFromFa	<i>Get transcript sequence from a GRangesList and a faFile or BSgenome</i>
--------------	--

---

### Description

For each GRanges object, find the sequence of it from faFile or BSgenome.

### Usage

```
txSeqsFromFa(grl, faFile, is.sorted = FALSE, keep.names = TRUE)
```

### Arguments

grl	a <a href="#">GRangesList</a> object
faFile	<a href="#">FaFile</a> , <a href="#">BSgenome</a> , fasta/index file path or an <a href="#">ORFik experiment</a> . This file is usually used to find the transcript sequences from some <a href="#">GRangesList</a> .
is.sorted	a speedup, if you know the grl ranges are sorted
keep.names	a logical, default (TRUE), if FALSE: return as character vector without names.

### Details

A small safety wrapper around [extractTranscriptSeqs](#) For debug of errors do: `which(!(unique(seqnamesPerGroup(grl, FALSE))` This happens usually when the grl contains chromosomes that the fasta file does not have. A normal error is that mitochondrial chromosome is called MT vs chrM even though they have same seqlevelsStyle. The above line will give you which chromosome it is missing.

### Value

a [DNAStringSet](#) of the transcript sequences

### See Also

Other [ExtendGenomicRanges](#): [asTX\(\)](#), [coveragePerTiling\(\)](#), [extendLeaders\(\)](#), [extendTrailers\(\)](#), [reduceKeepAttr\(\)](#), [tile1\(\)](#), [windowPerGroup\(\)](#)

---

uniqueGroups	<i>Get the unique set of groups in a GRangesList</i>
--------------	--

---

### Description

Sometimes [GRangesList](#) groups might be identical, for example ORFs from different isoforms can have identical ranges. Use this function to reduce these groups to unique elements in [GRangesList](#) grl, without names and metacolumns.

### Usage

```
uniqueGroups(grl)
```

**Arguments**

gr1                    a [GRangesList](#)

**Value**

a [GRangesList](#) of unique orfs

**See Also**

Other ORFHelpers: [defineTrailer\(\)](#), [longestORFs\(\)](#), [mapToGRanges\(\)](#), [orfID\(\)](#), [startCodons\(\)](#), [startSites\(\)](#), [stopCodons\(\)](#), [stopSites\(\)](#), [txNames\(\)](#), [uniqueOrder\(\)](#)

**Examples**

```
gr1 <- GRanges("1", IRanges(1,10), "+")
gr2 <- GRanges("1", IRanges(20, 30), "+")
# make a gr1 with duplicated ORFs (gr1 twice)
gr1 <- GRangesList(tx1_1 = gr1, tx2_1 = gr2, tx3_1 = gr1)
uniqueGroups(gr1)
```

---

uniqueOrder

*Get unique ordering for GRangesList groups*

---

**Description**

This function can be used to calculate unique numerical identifiers for each of the [GRangesList](#) elements. Elements of [GRangesList](#) are unique when the [GRanges](#) inside are not duplicated, so ranges differences matter as well as sorting of the ranges.

**Usage**

```
uniqueOrder(gr1)
```

**Arguments**

gr1                    a [GRangesList](#)

**Value**

an integer vector of indices of unique groups

**See Also**

[uniqueGroups](#)

Other ORFHelpers: [defineTrailer\(\)](#), [longestORFs\(\)](#), [mapToGRanges\(\)](#), [orfID\(\)](#), [startCodons\(\)](#), [startSites\(\)](#), [stopCodons\(\)](#), [stopSites\(\)](#), [txNames\(\)](#), [uniqueGroups\(\)](#)

## Examples

```
gr1 <- GRanges("1", IRanges(1,10), "+")
gr2 <- GRanges("1", IRanges(20, 30), "+")
# make a gr1 with duplicated ORFs (gr1 twice)
gr1 <- GRangesList(tx1_1 = gr1, tx2_1 = gr2, tx3_1 = gr1)
uniqueOrder(gr1) # remember ordering

# example on unique ORFs
uniqueORFs <- uniqueGroups(gr1)
# now the orfs are unique, let's map back to original set:
reMappedGr1 <- uniqueORFs[uniqueOrder(gr1)]
```

---

unlistGr1

*Safe unlist*

---

## Description

Same as [AnnotationDbi::unlist2()], keeps names correctly. Two differences is that if gr1 have no names, it will not make integer names, but keep them as null. Also if the GRangesList has names , and also the GRanges groups, then the GRanges group names will be kept.

## Usage

```
unlistGr1(gr1)
```

## Arguments

gr1                    a GRangesList

## Value

a GRanges object

## Examples

```
ORF <- GRanges(seqnames = "1",
               ranges = IRanges(start = c(1, 10, 20),
                                end = c(5, 15, 25)),
               strand = "+")
gr1 <- GRangesList(tx1_1 = ORF)
unlistGr1(gr1)
```

---

uORFSearchSpace      *Create search space to look for uORFs*

---

### Description

Given a GRangesList of 5' UTRs or transcripts, reassign the start sites using max peaks from CageSeq data (if CAGE is given). A max peak is defined as new TSS if it is within boundary of 5' leader range, specified by 'extension' in bp. A max peak must also be higher than minimum CageSeq peak cutoff specified in 'filterValue'. The new TSS will then be positioned where the cage read (with highest read count in the interval). If you want to include uORFs going into the CDS, add this argument too.

### Usage

```
uORFSearchSpace(
  fiveUTRs,
  cage = NULL,
  extension = 1000,
  filterValue = 1,
  restrictUpstreamToTx = FALSE,
  removeUnused = FALSE,
  cds = NULL
)
```

### Arguments

fiveUTRs	(GRangesList) The 5' leaders or full transcript sequences
cage	Either a filePath for the CageSeq file as .bed .bam or .wig, with possible compressions (".gzip", ".gz", ".bgz"), or already loaded CageSeq peak data as GRanges or GAlignment. NOTE: If it is a .bam file, it will add a score column by running: convertToOneBasedRanges(cage, method = "5prime", addScoreColumn = TRUE) The score column is then number of replicates of read, if score column is something else, like read length, set the score column to NULL first.
extension	The maximum number of bases upstream of the TSS to search for CageSeq peak.
filterValue	The minimum number of reads on cage position, for it to be counted as possible new tss. (represented in score column in CageSeq data) If you already filtered, set it to 0.
restrictUpstreamToTx	a logical (FALSE). If TRUE: restrict leaders to not extend closer than 5 bases from closest upstream leader, set this to TRUE.
removeUnused	logical (FALSE), if False: (standard is to set them to original annotation), If TRUE: remove leaders that did not have any cage support.
cds	(GRangesList) CDS of relative fiveUTRs, applicable only if you want to extend 5' leaders downstream of CDS's, to allow upstream ORFs that can overlap into CDS's.

### Value

a GRangesList of newly assigned TSS for fiveUTRs, using CageSeq data.



**See Also**

Other uorfs: [addCdsOnLeaderEnds\(\)](#), [filterUORFs\(\)](#), [removeORFsWithSameStartAsCDS\(\)](#), [removeORFsWithSameStartAsCDS\(\)](#), [removeORFsWithStartInsideCDS\(\)](#), [removeORFsWithinCDS\(\)](#)

**Examples**

```
# example 5' leader, notice exon_rank column
fiveUTRs <- GenomicRanges::GRangesList(
  GenomicRanges::GRanges(seqnames = "chr1",
    ranges = IRanges::IRanges(1000, 2000),
    strand = "+",
    exon_rank = 1))
names(fiveUTRs) <- "tx1"

# make fake CageSeq data from promoter of 5' leaders, notice score column
cage <- GenomicRanges::GRanges(
  seqnames = "chr1",
  ranges = IRanges::IRanges(500, 510),
  strand = "+",
  score = 10)

# finally reassign TSS for fiveUTRs
uORFSearchSpace(fiveUTRs, cage)
```

---

updateTxdbRanks

*Update exon ranks of exon data.frame inside txdb object*


---

**Description**

Update exon ranks of exon data.frame inside txdb object

**Usage**

```
updateTxdbRanks(exons)
```

**Arguments**

exons                    a data.frame, call of `as.list(txdb)$splicings`

**Value**

a data.frame, modified call of `as.list(txdb)`

---

updateTxdbStartSites *Update start sites of leaders*

---

### Description

Update start sites of leaders

### Usage

```
updateTxdbStartSites(txList, fiveUTRs, removeUnused)
```

### Arguments

txList	a list, call of <code>as.list(txdb)</code>
fiveUTRs	a <code>GRangesList</code> of 5' leaders
removeUnused	logical (FALSE), remove leaders that did not have any cage support. (standard is to set them to original annotation)

### Value

a list, modified call of `as.list(txdb)`

---

upstreamFromPerGroup *Get rest of objects upstream (inclusive)*

---

### Description

Per group get the part upstream of position. `upstreamFromPerGroup(tx, stopSites(fiveUTRs, asGR = TRUE))` will return the 5' utrs per transcript as `GRangesList`, usually used for interesting parts of the transcripts.

### Usage

```
upstreamFromPerGroup(tx, upstreamFrom)
```

### Arguments

tx	a <a href="#">GRangesList</a> , usually of Transcripts to be changed
upstreamFrom	a vector of integers, for each group in tx, where is the new start point of first valid exon.

### Details

If you don't want to include the points given in the region, use [upstreamOfPerGroup](#)

### Value

a `GRangesList` of upstream part

**See Also**

Other GRanges: [assignFirstExonsStartSite\(\)](#), [assignLastExonsStopSite\(\)](#), [downstreamFromPerGroup\(\)](#), [downstreamOfPerGroup\(\)](#), [upstreamOfPerGroup\(\)](#)

---

upstreamOfPerGroup     *Get rest of objects upstream (exclusive)*

---

**Description**

Per group get the part upstream of position `upstreamOfPerGroup(tx, startSites(cds, asGR = TRUE))` will return the 5' utrs per transcript, usually used for interesting parts of the transcripts.

**Usage**

```
upstreamOfPerGroup(tx, upstreamOf, allowOutside = TRUE)
```

**Arguments**

<code>tx</code>	a <a href="#">GRangesList</a> , usually of Transcripts to be changed
<code>upstreamOf</code>	a vector of integers, for each group in <code>tx</code> , where is the the base after the new stop point of last valid exon.
<code>allowOutside</code>	a logical (T), can <code>upstreamOf</code> extend outside range of <code>tx</code> , can set boundary as a false hit, so beware.

**Value**

a [GRangesList](#) of upstream part

**See Also**

Other GRanges: [assignFirstExonsStartSite\(\)](#), [assignLastExonsStopSite\(\)](#), [downstreamFromPerGroup\(\)](#), [downstreamOfPerGroup\(\)](#), [upstreamFromPerGroup\(\)](#)

---

validateExperiments     *Validate ORFik [experiment](#)*

---

**Description**

Check for valid existing, non-empty and all unique. A good way to see if your experiment is valid.

**Usage**

```
validateExperiments(df)
```

**Arguments**

<code>df</code>	an <a href="#">ORFik <code>experiment</code></a>
-----------------	--

**Value**

NULL (Stops if failed)

**See Also**

Other ORFik\_experiment: [ORFik.template.experiment\(\)](#), [bamVarName\(\)](#), [create.experiment\(\)](#), [experiment-class](#), [filepath\(\)](#), [libraryTypes\(\)](#), [organism.df\(\)](#), [outputLibs\(\)](#), [read.experiment\(\)](#), [save.experiment\(\)](#)

---

 validGRL

*Helper Function to check valid GRangesList input*


---

**Description**

Helper Function to check valid GRangesList input

**Usage**

```
validGRL(class, type = "grl", checkNULL = FALSE)
```

**Arguments**

class	as character vector the given class of supposed GRangesList object
type	a character vector, is it gtf, cds, 5', 3', for messages.
checkNULL	should NULL classes be checked and return indices of these?

**Value**

either NULL or indices (checkNULL == TRUE)

**See Also**

Other validity: [checkRFP\(\)](#), [checkRNA\(\)](#), [is.ORF\(\)](#), [is.gr\\_or\\_grl\(\)](#), [is.grl\(\)](#), [is.range\(\)](#), [validSeqlevels\(\)](#)

---

 validSeqlevels

*Helper function to find overlapping seqlevels*


---

**Description**

Keep only seqnames in reads that are in grl Useful to avoid seqname warnings in bioC

**Usage**

```
validSeqlevels(grl, reads)
```

**Arguments**

grl	a <a href="#">GRangesList</a> or GRanges object
reads	a GRanges, GAlignment or GAlignmentPairs object

**Value**

a character vector of valid seqlevels

**See Also**

Other validity: [checkRFP\(\)](#), [checkRNA\(\)](#), [is.ORF\(\)](#), [is.gr\\_or\\_grl\(\)](#), [is.grl\(\)](#), [is.range\(\)](#), [validGRL\(\)](#)

---

widthPerGroup	<i>Get list of widths per granges group</i>
---------------	---

---

**Description**

Get list of widths per granges group

**Usage**

```
widthPerGroup(grl, keep.names = TRUE)
```

**Arguments**

`grl` a [GRangesList](#)

`keep.names` a boolean, keep names or not, default: (TRUE)

**Value**

an integer vector (named/unnamed) of widths

**Examples**

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
  ranges = IRanges(c(7, 14), width = 3),
  strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
  ranges = IRanges(c(4, 1), c(9, 3)),
  strand = c("-", "-"))
grl <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
widthPerGroup(grl)
```

---

windowCoveragePlot     *Get meta coverage plot of reads*

---

### Description

Spanning a region like a transcripts, plot how the reads distribute.

### Usage

```

windowCoveragePlot(
  coverage,
  output = NULL,
  scoring = "zscore",
  colors = c("skyblue4", "orange"),
  title = "Coverage metaplot",
  type = "transcripts",
  scaleEqual = FALSE,
  setMinToZero = FALSE
)

```

### Arguments

coverage	a data.table, e.g. output of scaledWindowCoverage
output	character string (NULL), if set, saves the plot as pdf or png to path given. If no format is given, is save as pdf.
scoring	character vector, default "zscore", either of zscore, transcriptNormalized, sum, mean, median, .. or NULL. Set NULL if already scored. see ?coverageScorings for info and more alternatives.
colors	character vector colors to use in plot, will fix automatically, using binary splits with colors c('skyblue4', 'orange').
title	a character (metaplot) (what is the title of plot?)
type	a character (transcripts), what should legends say is the whole region? Transcripts, genes, non coding rnas etc.
scaleEqual	a logical (FALSE), should all fractions (rows), have same max value, for easy comparison of max values if needed.
setMinToZero	a logical (FALSE), should minimum y-value be 0 (TRUE). With FALSE minimum value is minimum score at any position. This parameter overrides scaleEqual.

### Details

If coverage has a column called feature, this can be used to subdivide the meta coverage into parts as (5' UTRs, cds, 3' UTRs) These are the columns in the plot. The fraction column divide sequence libraries. Like ribo-seq and rna-seq. These are the rows of the plot. If you return this function without assigning it and output is NULL, it will automatically plot the figure in your session. If output is assigned, no plot will be shown in session. NULL is returned and object is saved to output.

Colors: Remember if you want to change anything like colors, just return the ggplot object, and reassign like: obj + scale\_color\_brewer() etc.

**Value**

a ggplot object of the coverage plot, NULL if output is set, then the plot will only be saved to location.

**See Also**

Other coveragePlot: [coverageHeatMap\(\)](#), [pSitePlot\(\)](#), [savePlot\(\)](#)

**Examples**

```
library(data.table)
coverage <- data.table(position = seq(20),
                       score = sample(seq(20), 20, replace = TRUE))
windowCoveragePlot(coverage)

#Multiple plots in one frame:
coverage2 <- copy(coverage)
coverage$fraction <- "Ribo-seq"
coverage2$fraction <- "RNA-seq"
dt <- rbindlist(list(coverage, coverage2))
windowCoveragePlot(dt, scoring = "log10sum")

# See vignette for a more practical example
```

---

windowPerGroup	<i>Get window region of GRanges object</i>
----------------	--

---

**Description**

Per GRanges input (gr) of single position inputs, create a GRangesList window output of specified upstream, downstream region.  
 If downstream is 20, it means the window will start 20 downstream of gr start site (-20 in relative transcript coordinates.) If upstream is 20, it means the window will start 20 upstream of gr start site (+20 in relative transcript coordinates.) It will keep exon structure of tx, so if -20 is on next exon, it jumps to next exon.

**Usage**

```
windowPerGroup(gr, tx, upstream = 0L, downstream = 0L)
```

**Arguments**

gr	a GRanges/IRanges object (startSites or others, must be single point per in genomic coordinates)
tx	a <a href="#">GRangesList</a> of transcripts or (container region), names of tx must contain all gr names. The names of gr can also be the ORFik orf names. that is "tx-Name_id".
upstream	an integer, default (0), relative region to get upstream from.
downstream	an integer, default (0), relative region to get downstream from

**Details**

If a region has a part that goes out of bounds, E.g if you try to get window around the CDS start site, goes longer than the 5' leader start site, it will set start to the edge boundary (the TSS of the transcript in this case). If region has no hit in bound, a width 0 GRanges object is returned. This is usefull for things like countOverlaps, since 0 hits will then always be returned for the correct object index. If you don't want the 0 width windows, use reduce() to remove 0-width windows.

**Value**

a GRanges, or GRangesList object if any group had > 1 exon.

**See Also**

Other ExtendGenomicRanges: [asTX\(\)](#), [coveragePerTiling\(\)](#), [extendLeaders\(\)](#), [extendTrailers\(\)](#), [reduceKeepAttr\(\)](#), [tile1\(\)](#), [txSeqsFromFa\(\)](#)

**Examples**

```
# find 2nd codon of an ORF on a spliced transcript
ORF <- GRanges("1", c(3), "+") # start site
names(ORF) <- "tx1_1" # ORF 1 on tx1
tx <- GRangesList(tx1 = GRanges("1", c(1,3,5,7,9,11,13), "+"))
windowPerGroup(ORF, tx, upstream = -3, downstream = 5) # <- 2nd codon

# With multiple extensions downstream
ORF <- rep(ORF, 2)
names(ORF)[2] <- "tx1_2"
windowPerGroup(ORF, tx, upstream = 0, downstream = c(3, 5))
```

---

windowPerReadLength     *Find proportion of reads per position per read length in window*

---

**Description**

This is defined as: Fraction of reads per read length, per position in whole window (defined by upstream and downstream) If tx is not NULL, it gives a metaWindow, centered around startSite of grl from upstream and downstream. If tx is NULL, it will use only downstream, since it has no reference on how to find upstream region. The exception is when upstream is negative, that is, going into downstream region of the object.

**Usage**

```
windowPerReadLength(
  grl,
  tx = NULL,
  reads,
  pShifted = TRUE,
  upstream = if (pShifted) 5 else 20,
  downstream = if (pShifted) 20 else 5,
  acceptedLengths = NULL,
  zeroPosition = upstream,
```



```

    scoring = "transcriptNormalized",
    weight = "score"
  )

```

### Arguments

grl	a <a href="#">GRangesList</a> object with usually either leaders, cds', 3' utrs or ORFs
tx	default NULL, a <a href="#">GRangesList</a> of transcripts or (container region), names of tx must contain all grl names. The names of grl can also be the ORFik orf names. that is "txName_id"
reads	a <a href="#">GAlignments</a> or <a href="#">GRanges</a> object of RiboSeq, RnaSeq etc. Weights for scoring is default the 'score' column in 'reads'
pShifted	a logical (TRUE), are Ribo-seq reads p-shifted to size 1 width reads? If upstream and downstream is set, this argument is irrelevant. So set to FALSE if this is not p-shifted Ribo-seq.
upstream	an integer (5), relative region to get upstream from.
downstream	an integer (20), relative region to get downstream from
acceptedLengths	an integer vector (NULL), the read lengths accepted. Default NULL, means all lengths accepted.
zeroPosition	an integer DEFAULT (upstream), what is the center point? Like leaders and cds combination, then 0 is the TIS and -1 is last base in leader. NOTE!: if windows have different widths, this will be ignored.
scoring	a character (transcriptNormalized), which meta coverage scoring ? one of (zscore, transcriptNormalized, mean, median, sum, sumLength, fracPos), see ?coverageScorings for more info. Use to decide a scoring of hits per position for metacoverage etc. Set to NULL if you do not want meta coverage, but instead want per gene per position raw counts.
weight	(default: 'score'), if defined a character name of valid meta column in subject. <a href="#">GRanges("chr1", 1, "+", score = 5)</a> , would mean score column tells that this alignment region was found 5 times. ORFik .bedo files, contains a score column like this. As do CAGEr CAGE files and many other package formats. You can also assign a score column manually.

### Value

a data.table with lengths by coverage / vector of proportions

### See Also

Other coverage: [coverageScorings\(\)](#), [metaWindow\(\)](#), [regionPerReadLength\(\)](#), [scaledWindowPositions\(\)](#)

### Examples

```

cds <- GRangesList(tx1 = GRanges("1", 100:129, "+"))
tx <- GRangesList(tx1 = GRanges("1", 80:129, "+"))
reads <- GRanges("1", seq(79,129, 3), "+")
windowPerReadLength(cds, tx, reads, scoring = "sum")
windowPerReadLength(cds, tx, reads, scoring = "transcriptNormalized")

```

---

windowPerTranscript     *Get a binned coverage window per transcript*

---

### Description

Per transcript (or other regions), bin them all to windowSize (default 100), and make a data.table, rows are positions, useful for plotting with ORFik and ggplot2.

### Usage

```

windowPerTranscript(
  txdb,
  reads,
  splitIn3 = TRUE,
  windowSize = 100,
  fraction = "1",
  weight = "score",
  BPPARAM = bpparam()
)

```

### Arguments

txdb	a TxDb object or a path to gtf/gff/db file.
reads	GRanges or GAlignment of reads
splitIn3	a logical(TRUE), split window in 3 (leader, cds, trailer)
windowSize	an integer (100), size of windows (columns)
fraction	a character (1), info on reads (which read length, or which type (RNA seq)) (row names)
weight	(default: 'score'), if defined a character name of valid meta column in subject. GRanges("chr1", 1, "+", score = 5), would mean score column tells that this alignment region was found 5 times. ORFik .bedo files, contains a score column like this. As do CAGEr CAGE files and many other package formats. You can also assign a score column manually.
BPPARAM	how many cores/threads to use? default: bpparam()

### Details

NOTE: All ranges with smaller width than windowSize, will of course be removed. What is the 100th position on a 1 width object ?

### Value

a data.table with columns position, score

---

xAxisScaler	<i>Scale x axis correctly</i>
-------------	-------------------------------

---

**Description**

Works for all coverage plots, that need 0 position aligning

**Usage**

```
xAxisScaler(covPos)
```

**Arguments**

covPos            a numeric vector of positions in coverage

**Details**

It basically bins the x axis on  $\text{floor}(\text{length of x axis} / 20)$  or 1 if  $x < 20$

**Value**

a numeric vector from the seq() function, aligned to 0.

---

yAxisScaler	<i>Scale y axis correctly</i>
-------------	-------------------------------

---

**Description**

Works for all coverage plots.

**Usage**

```
yAxisScaler(covPos, increments.y = "auto")
```

**Arguments**

covPos            a levels object from a factor of y axis  
 increments.y    increments of y axis, default "auto". Or a numeric value  $<$  max position  $\&$   $>$  min position.

**Value**

a character vector from the seq() function, aligned to 0.

# Index

- \* **CAGE**
  - assignTSSByCage, 13
  - reassignTSSbyCage, 164
  - reassignTxDbByCage, 166
- \* **ExtendGenomicRanges**
  - asTX, 15
  - coveragePerTiling, 40
  - extendLeaders, 69
  - extendTrailers, 70
  - reduceKeepAttr, 167
  - tile1, 217
  - txSeqsFromFa, 229
  - windowPerGroup, 239
- \* **GRanges**
  - assignFirstExonsStartSite, 12
  - assignLastExonsStopSite, 13
  - downstreamFromPerGroup, 54
  - downstreamOfPerGroup, 55
  - upstreamFromPerGroup, 234
  - upstreamOfPerGroup, 235
- \* **ORFHelpers**
  - defineTrailer, 46
  - longestORFs, 134
  - mapToGRanges, 137
  - orfID, 144
  - startCodons, 203
  - startSites, 207
  - stopCodons, 208
  - stopSites, 211
  - txNames, 227
  - uniqueGroups, 229
  - uniqueOrder, 230
- \* **ORFik\_experiment**
  - bamVarName, 16
  - create.experiment, 43
  - experiment-class, 59
  - filepath, 71
  - libraryTypes, 129
  - ORFik.template.experiment, 144
  - organism.df, 147
  - outputLibs, 148
  - read.experiment, 160
  - save.experiment, 180
  - validateExperiments, 235
- \* **QC report**
  - QCplots, 155
  - QCreport, 156
  - QCstats, 157
- \* **STAR**
  - getGenomeAndAnnotation, 96
  - install.fastp, 118
  - STAR.align.folder, 193
  - STAR.align.single, 197
  - STAR.allsteps.multiQC, 199
  - STAR.index, 200
  - STAR.install, 201
  - STAR.multiQC, 202
  - STAR.remove.crashed.genome, 203
- \* **TE**
  - DTEG.analysis, 56
  - DTEG.plot, 57
  - te.table, 215
  - te\_rna.plot, 216
- \* **countTable**
  - countTable, 35
  - countTable\_regions, 36
- \* **coveragePlot**
  - coverageHeatMap, 38
  - pSitePlot, 154
  - savePlot, 181
  - windowCoveragePlot, 238
- \* **coverage**
  - coverageScorings, 42
  - metaWindow, 140
  - regionPerReadLength, 168
  - scaledWindowPositions, 182
  - windowPerReadLength, 240
- \* **experiment plots**
  - transcriptWindow, 221
  - transcriptWindow1, 222
  - transcriptWindowPer, 223
- \* **experiment\_naming**
  - cellLineNames, 18
  - conditionNames, 30
  - libNames, 129
  - mainNames, 134

- repNames, 175
- stageNames, 193
- tissueNames, 218
- \* **features**
  - computeFeatures, 26
  - computeFeaturesCage, 28
  - countOverlapsW, 34
  - disengagementScore, 49
  - distToCds, 50
  - distToTSS, 51
  - entropy, 58
  - floss, 88
  - fpkm, 90
  - fpkm\_calc, 92
  - fractionLength, 92
  - initiationScore, 115
  - insideOutsideORF, 116
  - isInFrame, 121
  - isOverlapping, 122
  - kozakSequenceScore, 125
  - orfScore, 146
  - rankOrder, 159
  - ribosomeReleaseScore, 177
  - ribosomeStallingScore, 179
  - startRegion, 205
  - startRegionCoverage, 206
  - stopRegion, 210
  - subsetCoverage, 213
  - translationalEff, 224
- \* **findORFs**
  - findMapORFs, 78
  - findORFs, 81
  - findORFsFasta, 82
  - findUORFs, 84
  - startDefinition, 204
  - stopDefinition, 209
- \* **heatmaps**
  - coverageHeatMap, 38
  - heatMap\_single, 111
  - heatMapL, 108
  - heatMapRegion, 109
- \* **pshifting**
  - changePointAnalysis, 19
  - detectRibosomeShifts, 47
  - shiftFootprints, 184
  - shiftFootprintsByExperiment, 186
- \* **uorfs**
  - addCdsOnLeaderEnds, 8
  - filterUORFs, 75
  - removeORFsWithinCDS, 172
  - removeORFsWithSameStartAsCDS, 172
  - removeORFsWithSameStopAsCDS, 173
- removeORFsWithStartInsideCDS, 173
- uORFSearchSpace, 232
- \* **utils**
  - bedToGR, 18
  - convertToOneBasedRanges, 32
  - export.bed12, 62
  - export.wiggle, 68
  - fimport, 75
  - findFa, 76
  - fread.bed, 93
  - optimizeReads, 143
  - readBam, 161
  - readWig, 163
- \* **validity**
  - checkRFP, 20
  - checkRNA, 20
  - is.gr\_or\_grl, 120
  - is.grl, 119
  - is.ORF, 120
  - is.range, 121
  - validGRL, 236
  - validSeqlevels, 236
- addCdsOnLeaderEnds, 8, 75, 172, 173, 233
- addNewTSSOnLeaders, 9
- allFeaturesHelper, 9
- artificial.orfs, 10
- assignAnnotations, 11
- assignFirstExonsStartSite, 12, 13, 54, 55, 235
- assignLastExonsStopSite, 12, 13, 54, 55, 235
- assignTSSByCage, 13, 165, 167
- asTX, 15, 41, 69, 71, 168, 217, 229, 240
- bamVarName, 16, 44, 60, 72, 130, 145, 148, 149, 160, 181, 236
- bamVarNamePicker, 17
- bedToGR, 18, 33, 62, 68, 76, 77, 94, 144, 162, 164
- cellLineNames, 18, 30, 129, 135, 175, 193, 218
- changePointAnalysis, 19, 48, 185, 187
- checkRFP, 20, 20, 120, 121, 236, 237
- checkRNA, 20, 20, 120, 121, 236, 237
- codonSumsPerGroup, 21
- collapse.by.scores, 21
- collapse.fastq, 22
- collapseDuplicatedReads, 23
- collapseDuplicatedReads,GAlignmentPairs-method, 23

- collapseDuplicatedReads, GAlignments-method, 24
- collapseDuplicatedReads, GRanges-method, 25
- combn.pairs, 26
- computeFeatures, 26, 29, 35, 50–52, 59, 89, 91–93, 116, 117, 122, 126, 147, 159, 178, 179, 205, 207, 210, 213, 226
- computeFeaturesCage, 27, 28, 35, 50–52, 59, 89, 91–93, 116, 117, 122, 126, 147, 159, 178, 179, 205, 207, 210, 213, 226
- conditionNames, 18, 30, 129, 135, 175, 193, 218
- convertLibs, 30
- convertToOneBasedRanges, 18, 31, 32, 62, 68, 76, 77, 94, 144, 162, 164, 190
- correlation.plots, 33
- countOverlapsW, 27, 29, 34, 50–52, 59, 89, 91–93, 116, 117, 122, 126, 147, 159, 178, 179, 205, 207, 210, 213, 226
- countTable, 35, 37, 145, 156
- countTable\_regions, 36, 36
- coverageByTranscript, 37
- coverageByTranscriptW, 37
- coverageGroupings, 38
- coverageHeatMap, 38, 109, 110, 112, 155, 181, 239
- coveragePerTiling, 16, 40, 69, 71, 168, 217, 229, 240
- coverageScorings, 42, 141, 169, 182, 241
- create.experiment, 16, 43, 60, 72, 130, 145, 148, 149, 160, 181, 236
- data.frame, 18
- defineIsoform, 45
- defineTrailer, 46, 134, 138, 144, 204, 208, 211, 228, 230
- detectRibosomeShifts, 19, 47, 146, 185–187
- disengagementScore, 27, 29, 35, 49, 51, 52, 59, 89, 91–93, 116, 117, 122, 126, 147, 159, 178, 179, 205, 207, 210, 213, 226
- distToCds, 27, 29, 35, 50, 50, 52, 59, 89, 91–93, 116, 117, 122, 126, 147, 159, 178, 179, 205, 207, 210, 213, 226
- distToTSS, 27, 29, 35, 50, 51, 51, 59, 89, 91–93, 116, 117, 122, 126, 147, 159, 178, 179, 205, 207, 210, 213, 226
- DNAStrngSet, 229
- download.SRA, 52
- download.SRA.metadata, 53
- downstreamFromPerGroup, 12, 13, 54, 55, 235
- downstreamN, 55
- downstreamOfPerGroup, 12, 13, 54, 55, 235
- DTEG.analysis, 56, 57, 58, 215, 216
- DTEG.plot, 57, 57, 215, 216
- entropy, 27, 29, 35, 50–52, 58, 89, 91–93, 116, 117, 122, 126, 147, 159, 178, 179, 205, 207, 210, 213, 226
- experiment, 10, 16, 17, 27, 28, 31, 34, 35, 37, 43, 56, 61, 71, 76, 77, 108, 110, 125, 126, 129, 136, 142, 145, 148, 149, 151, 155–158, 160, 162, 170, 177, 180, 186, 188–190, 207, 214, 215, 221–224, 227, 229, 235
- experiment (experiment-class), 59
- experiment-class, 59
- experiment.colors, 61, 151, 221, 223
- export.bed12, 18, 33, 62, 68, 76, 77, 94, 144, 162, 164
- export.bedo, 31, 63, 191
- export.bedoc, 31, 63, 187, 191
- export.ofst, 64
- export.ofst, GAlignmentPairs-method, 65
- export.ofst, GAlignments-method, 66
- export.ofst, GRanges-method, 67
- export.wiggle, 18, 33, 62, 68, 76, 77, 94, 144, 162, 164, 187
- extendLeaders, 16, 41, 69, 71, 168, 217, 229, 240
- extendsTSSexons, 70
- extendTrailers, 16, 41, 69, 70, 168, 217, 229, 240
- extractTranscriptSeqs, 229
- FaFile, 10, 27, 28, 76, 78, 81, 85, 125, 126, 207, 229
- filepath, 16, 44, 60, 71, 130, 145, 148, 149, 160, 181, 236
- filterCage, 72
- filterExtremePeakGenes, 73
- filterTranscripts, 74
- filterUORFs, 8, 75, 172, 173, 233
- fimport, 18, 33, 62, 68, 75, 77, 94, 144, 162, 164
- findFa, 18, 33, 62, 68, 76, 76, 94, 144, 162, 164
- findFromPath, 77
- findLibrariesInFolder, 77
- findMapORFs, 78, 82, 83, 86, 204, 209
- findMaxPeaks, 79
- findNewTSS, 80

- findNGSPairs, 80
- findORFs, 79, 81, 83, 86, 204, 209
- findORFsFasta, 79, 82, 82, 86, 204, 209
- findPeaksPerGene, 83
- findUORFs, 79, 82, 83, 84, 204, 209
- firstEndPerGroup, 86
- firstExonPerGroup, 87
- firstStartPerGroup, 87
- floss, 27, 29, 35, 50–52, 59, 88, 91–93, 116, 117, 122, 126, 147, 159, 178, 179, 205, 207, 210, 213, 226
- footprints.analysis, 90
- fpkm, 27, 29, 35, 50–52, 59, 89, 90, 92, 93, 116, 117, 122, 126, 147, 159, 178, 179, 205, 207, 210, 213, 226
- fpkm\_calc, 27, 29, 35, 50–52, 59, 89, 91, 92, 93, 116, 117, 122, 126, 147, 159, 178, 179, 205, 207, 210, 213, 226
- fractionLength, 27, 29, 35, 50–52, 59, 89, 91, 92, 92, 116, 117, 122, 126, 147, 159, 178, 179, 205, 207, 210, 213, 226
- fread.bed, 18, 33, 62, 68, 76, 77, 93, 144, 162, 164
  
- GAlignmentPairs, 114, 161
- GAlignments, 10, 27, 28, 37, 41, 47, 58, 76, 88, 90, 110, 111, 115, 146, 161, 169, 178, 185, 225, 241
- GappedReads, 161
- gcContent, 94
- get\_genome\_fasta, 100
- get\_genome\_gtf, 101
- get\_noncoding\_rna, 102
- get\_phix\_genome, 104
- get\_silva\_rRNA, 105
- getGAlignments, 95
- getGAlignmentsPairs, 95
- getGenomeAndAnnotation, 96, 118, 196, 199–203
- getGRanges, 98
- getNGenesCoverage, 99
- getWeights, 27, 99, 115, 147
- GRanges, 10, 18, 27, 28, 37, 41, 58, 76, 88, 90, 94, 110, 111, 146, 164, 169, 178, 185, 225, 230, 241
- GRangesList, 10, 12, 13, 15, 27, 28, 37, 41, 49–51, 54, 55, 58, 69, 70, 78, 86–88, 90, 93, 107, 108, 111, 115, 117, 125, 127, 128, 134–138, 142–144, 146, 159, 167, 169, 176, 178–180, 184, 191, 192, 203, 205–208, 210–212, 217, 221, 224, 225, 227, 229, 230, 234–237, 239, 241
- groupGRangesBy, 105
- groupings, 106
- gSort, 107
  
- hasHits, 107
- heatMap\_single, 40, 109, 110, 111
- heatMapL, 40, 108, 110, 112
- heatMapRegion, 40, 109, 109, 112
  
- import.bed, 93
- import.bedo, 112
- import.bedoc, 113
- import.ofst, 114
- importGtfFromTxdb, 115
- initiationScore, 27, 29, 35, 50–52, 59, 89, 91–93, 115, 117, 122, 126, 147, 159, 178, 179, 205, 207, 210, 213, 226
- insideOutsideORF, 27, 29, 35, 50–52, 59, 89, 91–93, 116, 116, 122, 126, 147, 159, 178, 179, 205, 207, 210, 213, 226
- install.fastp, 98, 101–104, 118, 196, 199–203
- install.sratoolkit, 119
- IRanges, 81
- IRangesList, 81
- is.gr\_or\_grl, 20, 120, 120, 121, 236, 237
- is.grl, 20, 119, 120, 121, 236, 237
- is.ORF, 20, 120, 120, 121, 236, 237
- is.range, 20, 120, 121, 236, 237
- isInFrame, 27, 29, 35, 50–52, 59, 89, 91–93, 116, 117, 121, 122, 126, 147, 159, 178, 179, 205, 207, 210, 213, 226
- isOverlapping, 27, 29, 35, 50–52, 59, 89, 91–93, 116, 117, 122, 122, 126, 147, 159, 178, 179, 205, 207, 210, 213, 226
- isPeriodic, 48, 123
  
- kozak\_IR\_ranking, 126
- kozakHeatmap, 123
- kozakSequenceScore, 27, 29, 35, 50–52, 59, 89, 91–93, 116, 117, 122, 125, 147, 159, 178, 179, 205, 207, 210, 213, 226
  
- lastExonEndPerGroup, 127
- lastExonPerGroup, 128
- lastExonStartPerGroup, 128
- libNames, 18, 30, 129, 135, 175, 193, 218
- libraryTypes, 16, 44, 60, 72, 129, 145, 148, 149, 160, 181, 236

- list.experiments, 130
- loadRegion, 131
- loadRegions, 131
- loadTranscriptType, 132
- loadTxdb, 133
- longestORFs, 46, 78, 81, 83, 85, 134, 138, 144, 204, 208, 211, 228, 230
- mainNames, 18, 30, 129, 134, 175, 193, 218
- makeExonRanks, 135
- makeORFNames, 135
- makeSummarizedExperimentFromBam, 136
- mapToGRanges, 46, 134, 137, 144, 204, 208, 211, 228, 230
- matchColors, 138
- matchNaming, 138
- matchSeqStyle, 139
- mergeFastq, 139
- metaWindow, 43, 140, 169, 182, 241
- nrow, experiment-method, 142
- numCodons, 142
- numExonsPerGroup, 143
- optimizeReads, 18, 33, 62, 68, 76, 77, 94, 143, 162, 164
- orfID, 46, 134, 138, 144, 204, 208, 211, 228, 230
- ORFik (ORFik-package), 7
- ORFik-package, 7
- ORFik.template.experiment, 16, 44, 60, 72, 130, 144, 148, 149, 160, 181, 236
- ORFikQC, 35, 145
- orfScore, 27, 29, 35, 50–52, 59, 89, 91–93, 116, 117, 122, 126, 146, 159, 178, 179, 205, 207, 210, 213, 226
- organism.df, 16, 44, 60, 72, 130, 145, 147, 149, 160, 181, 236
- outputLibs, 16, 44, 60, 72, 130, 145, 148, 148, 160, 181, 236
- pasteDir, 149
- percentage\_to\_ratio, 150
- plotHelper, 150
- pmapFromTranscriptF, 151
- pmapToTranscriptF, 152
- prettyScoring, 153
- pseudo.transform, 153
- pSitePlot, 40, 154, 181, 239
- QC\_count\_tables, 158
- QCplots, 146, 155, 156, 157
- QCreport, 155, 156, 157
- QCstats, 145, 146, 155, 156, 157
- QCstats.plot, 157
- rankOrder, 27, 29, 35, 50–52, 59, 89, 91–93, 116, 117, 122, 126, 147, 159, 178, 179, 205, 207, 210, 213, 226
- read.experiment, 16, 44, 60, 72, 130, 145, 148, 149, 160, 181, 236
- readBam, 18, 33, 62, 68, 76, 77, 94, 144, 161, 164
- readGAlignments, 161
- readLengthTable, 162
- readWidths, 162
- readWig, 18, 33, 62, 68, 76, 77, 94, 144, 162, 163
- reassignTSSbyCage, 14, 164, 167
- reassignTxDbByCage, 14, 165, 166
- reduce, 168
- reduceKeepAttr, 16, 41, 69, 71, 167, 217, 229, 240
- regionPerReadLength, 43, 141, 168, 182, 241
- remakeTxdbExonIds, 170
- remove.experiments, 170
- remove.file\_ext, 171
- removeMetaCols, 171
- removeORFsWithinCDS, 8, 75, 172, 172, 173, 233
- removeORFsWithSameStartAsCDS, 8, 75, 172, 172, 173, 233
- removeORFsWithSameStopAsCDS, 8, 75, 172, 173, 173, 233
- removeORFsWithStartInsideCDS, 8, 75, 172, 173, 173, 233
- removeTxdbExons, 174
- removeTxdbTranscripts, 174
- rename.SRA.files, 175
- repNames, 18, 30, 129, 135, 175, 193, 218
- restrictTSSByUpstreamLeader, 176
- reverseMinusStrandPerGroup, 176
- RiboQC.plot, 177
- ribosomeReleaseScore, 27, 29, 35, 50–52, 59, 89, 91–93, 116, 117, 122, 126, 147, 159, 177, 179, 205, 207, 210, 213, 226
- ribosomeStallingScore, 27, 29, 35, 50–52, 59, 89, 91–93, 116, 117, 122, 126, 147, 159, 178, 179, 205, 207, 210, 213, 226
- rnaNormalize, 180
- save.experiment, 16, 44, 60, 72, 130, 145, 148, 149, 160, 180, 236



- savePlot, [40](#), [155](#), [181](#), [239](#)
- scaledWindowPositions, [43](#), [141](#), [169](#), [182](#), [241](#)
- scanBam, [161](#)
- ScanBamParam, [161](#)
- scoreSummarizedExperiment, [183](#)
- seqlevelsStyle, [76](#), [94](#), [133](#), [139](#), [149](#), [161](#), [164](#)
- seqnamesPerGroup, [184](#)
- shiftFootprints, [19](#), [48](#), [184](#), [187](#)
- shiftFootprintsByExperiment, [19](#), [48](#), [185](#), [186](#)
- shiftPlots, [188](#)
- shifts.load, [189](#)
- show, experiment-method, [189](#)
- simpleLibs, [190](#)
- sort.GenomicRanges, [191](#)
- sortPerGroup, [69](#), [70](#), [191](#)
- splitIn3Tx, [192](#)
- stageNames, [18](#), [30](#), [129](#), [135](#), [175](#), [193](#), [218](#)
- STAR.align.folder, [98](#), [101–104](#), [118](#), [193](#), [199–203](#)
- STAR.align.single, [98](#), [101–104](#), [118](#), [196](#), [197](#), [200–203](#)
- STAR.allsteps.multiQC, [98](#), [101–104](#), [118](#), [196](#), [199](#), [199](#), [201–203](#)
- STAR.index, [98](#), [101–104](#), [118](#), [196](#), [199](#), [200](#), [200](#), [202](#), [203](#)
- STAR.install, [98](#), [101–104](#), [118](#), [196](#), [199–201](#), [201](#), [202](#), [203](#)
- STAR.multiQC, [98](#), [101–104](#), [118](#), [196](#), [199–202](#), [202](#), [203](#)
- STAR.remove.crashed.genome, [98](#), [101–104](#), [118](#), [196](#), [199–202](#), [203](#)
- startCodons, [46](#), [134](#), [138](#), [144](#), [203](#), [205](#), [208](#), [211](#), [228](#), [230](#)
- startDefinition, [78](#), [79](#), [81–83](#), [85](#), [86](#), [204](#), [209](#)
- startRegion, [27](#), [29](#), [35](#), [50–52](#), [59](#), [89](#), [91–93](#), [116](#), [117](#), [122](#), [126](#), [147](#), [159](#), [178](#), [179](#), [205](#), [207](#), [210](#), [213](#), [226](#)
- startRegionCoverage, [27](#), [29](#), [35](#), [50–52](#), [59](#), [89](#), [91–93](#), [116](#), [117](#), [122](#), [126](#), [147](#), [159](#), [178](#), [179](#), [205](#), [206](#), [210](#), [213](#), [226](#)
- startRegionString, [207](#)
- startSites, [46](#), [134](#), [138](#), [144](#), [204](#), [207](#), [208](#), [211](#), [228](#), [230](#)
- stopCodons, [46](#), [134](#), [138](#), [144](#), [204](#), [208](#), [208](#), [210](#), [211](#), [228](#), [230](#)
- stopDefinition, [78](#), [79](#), [81–83](#), [85](#), [86](#), [204](#), [209](#)
- stopRegion, [27](#), [29](#), [35](#), [50–52](#), [59](#), [89](#), [91–93](#), [116](#), [117](#), [122](#), [126](#), [147](#), [159](#), [178](#), [179](#), [205](#), [207](#), [210](#), [213](#), [226](#)
- stopSites, [46](#), [134](#), [138](#), [144](#), [204](#), [208](#), [211](#), [228](#), [230](#)
- strandBool, [211](#)
- strandPerGroup, [212](#)
- subsetCoverage, [27](#), [29](#), [35](#), [50–52](#), [59](#), [89](#), [91–93](#), [116](#), [117](#), [122](#), [126](#), [147](#), [159](#), [178](#), [179](#), [205](#), [207](#), [210](#), [213](#), [226](#)
- subsetToFrame, [213](#)
- SummarizedExperiment, [59](#), [137](#), [145](#), [156](#)
- te.plot, [214](#)
- te.table, [57](#), [58](#), [215](#), [216](#)
- te\_rna.plot, [57](#), [58](#), [215](#), [216](#)
- tile1, [16](#), [41](#), [69](#), [71](#), [168](#), [217](#), [229](#), [240](#)
- tissueNames, [18](#), [30](#), [129](#), [135](#), [175](#), [193](#), [218](#)
- TOP.Motif.ecdf, [218](#)
- topMotif, [220](#)
- transcriptWindow, [221](#), [223](#), [224](#)
- transcriptWindow1, [222](#), [222](#), [224](#)
- transcriptWindowPer, [222](#), [223](#), [223](#)
- translationalEff, [27](#), [29](#), [35](#), [50–52](#), [59](#), [89](#), [91–93](#), [116](#), [117](#), [122](#), [126](#), [147](#), [159](#), [178](#), [179](#), [205](#), [207](#), [210](#), [213](#), [224](#)
- trim\_detection, [227](#)
- trimming.table, [226](#)
- TxDB, [49](#), [117](#)
- txNames, [46](#), [134](#), [138](#), [144](#), [204](#), [208](#), [211](#), [227](#), [230](#)
- txNamesToGeneNames, [228](#)
- txSeqsFromFa, [16](#), [41](#), [69](#), [71](#), [168](#), [217](#), [229](#), [240](#)
- uniqueGroups, [46](#), [134](#), [138](#), [144](#), [204](#), [208](#), [211](#), [228](#), [229](#), [230](#)
- uniqueOrder, [46](#), [134](#), [138](#), [144](#), [204](#), [208](#), [211](#), [228](#), [230](#), [230](#)
- unlistGrl, [231](#)
- uORFSearchSpace, [8](#), [75](#), [172](#), [173](#), [232](#)
- updateTxdbRanks, [233](#)
- updateTxdbStartSites, [234](#)
- upstreamFromPerGroup, [12](#), [13](#), [54](#), [55](#), [234](#), [235](#)
- upstreamOfPerGroup, [12](#), [13](#), [54](#), [55](#), [234](#), [235](#), [235](#)
- validateExperiments, [16](#), [44](#), [60](#), [72](#), [130](#), [145](#), [148](#), [149](#), [160](#), [181](#), [235](#)
- validGRL, [20](#), [120](#), [121](#), [236](#), [237](#)
- validSeqlevels, [20](#), [120](#), [121](#), [236](#), [236](#)
- widthPerGroup, [237](#)

windowCoveragePlot, [40](#), [155](#), [181](#), [238](#)  
windowPerGroup, [16](#), [41](#), [69](#), [71](#), [168](#), [217](#),  
[229](#), [239](#)  
windowPerReadLength, [43](#), [141](#), [169](#), [182](#),  
[240](#)  
windowPerTranscript, [242](#)  
  
xAxisScaler, [243](#)  
  
yAxisScaler, [243](#)