# Package 'annmap'

March 19, 2019

**Type** Package

**Title** Genome annotation and visualisation package pertaining to Affymetrix arrays and NGS analysis.

**Description** annmap provides annotation mappings for Affymetrix exon arrays and coordinate based queries to support deep sequencing data analysis. Database access is hidden behind the API which provides a set of functions such as genesInRange(), geneToExon(), exonDetails(), etc. Functions to plot gene architecture and BAM file data are also provided. Underlying data are from Ensembl.

**Version** 1.25.0

**Date** 2011-09-14

**Author** Tim Yates <Tim.Yates@cruk.manchester.ac.uk>

**Maintainer** Chris Wirth <Christopher.Wirth@cruk.manchester.ac.uk>

**Depends** R (>= 2.15.0), methods, GenomicRanges

**Imports** DBI, RMySQL (>= 0.6-0), digest, Biobase, grid, lattice, Rsamtools, genefilter, IRanges, BiocGenerics

**Suggests** RUnit, rjson, Gviz

**License** GPL-2

**URL** http://annmap.cruk.manchester.ac.uk

**Collate** zzz.R db.R utils.R cache.R statements.R filtering.R utr.R coords.R ws.R plot.genomic.R plot.ngs.R si.R

**biocViews** Annotation, Microarray, OneChannel, ReportWriting, Transcription, Visualization

**git_url** https://git.bioconductor.org/packages/annmap

**git_branch** master

**git_last_commit** cfdf55a

**git_last_commit_date** 2018-10-30

**Date/Publication** 2019-03-19

# R topics documented:



---

`annmap-package` *Provide access to the Annmap annotation database*

---

## Description

Annmap `http://annmap.cruk.manchester.ac.uk` Is a genome annotation database and genome browser, based on the Google Maps API. The underlying annotation is derived from ENSEMBL (`http://www.ensembl.org`). Annmap also provides probe to genome mappings for Affymetrix Exon, Gene and Plus2 arrays.

The annmap package makes the data in annmap available for use within R and BioConductor.

## Details

| | |
|---|---|
| Package: | annmap |
| Type: | Package |
| Version: | 1.0.0 |
| Date: | 2011-09-14 |
| License: | GPL-2 |

## Author(s)

Tim Yates
Maintainer: Tim Yates <tyates@picr.man.ac.uk>

## References

Yates T, Okoniewski MJ, Miller CJ. X:Map: annotation and visualization of genome structure for Affymetrix exon array analysis. Nucleic Acids Res. 2008 Jan;36(Database issue):D780-6. Epub

2007 Oct 11.
http://nar.oxfordjournals.org/cgi/content/full/gkm779v1

## See Also

GenomicRanges

---

annmapAll *annmap 'all' functions*

---

## Description

Get all annotations for a given feature. For example, `allGenes` will return data for all the genes in the genome.

## Usage

```
allArrays( as.vector=FALSE )
allChromosomes( as.vector=FALSE )
allDomains( as.vector=FALSE )
allEstExons( as.vector=FALSE )
allEstGenes( as.vector=FALSE )
allEstTranscripts( as.vector=FALSE )
allExons( as.vector=FALSE )
allGenes( as.vector=FALSE )
allPredictionTranscripts( as.vector=FALSE )
allProbes( as.vector=FALSE )
allProbesets( as.vector=FALSE )
allProteins( as.vector=FALSE )
allSymbols( as.vector=FALSE )
allSynonyms( as.vector=FALSE )
allTranscripts( as.vector=FALSE )
```

## Arguments

| | |
|---|---|
| `as.vector` | If `TRUE` returns a vector of database identifiers. If `FALSE` returns a `GRanges` object containing detailed annotation. |

## Value

Returns a `vector` or `GRanges` object, as defined by `as.vector`.

## Author(s)

Tim Yates

## See Also

annmapTo
annmapDetails
annmapRange
annmapUtils
annmapFilters
GRanges

## Examples

```
if(interactive()) {
  annmapConnect()
  allChromosomes()
  allChromosomes(as.vector=TRUE)
}
```

---

annmapCoords *annmap co-ordinate mapping functions*

---

## Description

Functions to go between Genomic, Proteomic and Transcriptual co-ordinate systems.

## Usage

```
transcriptCoordsToGenome( transcript.ids, position=1, as.vector=FALSE, check.bounds=TRUE, trunca
genomeToTranscriptCoords( position, transcript.ids, as.vector=FALSE, check.bounds=TRUE, end=c( '
proteinCoordsToGenome( protein.ids, position=1, as.vector=FALSE, check.bounds=TRUE, truncate=TRU
genomeToProteinCoords( position, protein.ids, as.vector=FALSE, check.bounds=TRUE )
```

## Arguments

| | |
|---|---|
| transcript.ids | A vector of transcript.ids (or a RangedData object of transcripts returned from another annmap function) |
| position | The position of interest (either a genomic position for both of the genomeToXXXX methods, or a protein or transcript sequence position for the other two methods ) |
| as.vector | Should the returned data be in the form of a vector (if TRUE) or a RangedData object (if FALSE) |
| check.bounds | If TRUE, any postion outside the range of the protein/transcript will cause a warning to be issued and NA returned. |
| end | Should the UTR be taken in to account when calculating the location, one of ("none", "both", "3" or "5"). Defaults to none. |
| truncate | If truncate=TRUE, any lengths beyond the end of the transcript or protein will be set to the last residue |
| cds | If cds=TRUE then only the coding exons (or sub-regions of exons that are coding) are taken in to account. |
| protein.ids | A vector of protein.ids (or a RangedData object of proteins returned from another annmap function) |

## Details

The mapping functions need to deal with mappings that fall outside a transcript or protein (or within an intron). When as.vector=FALSE these are identified as NA in the results. Since RangedData objects cannot represent NA or missing values, when as.vector=FALSE, all locations which cannot be mapped are dropped from the result.

## Author(s)

Tim Yates

## See Also

annmapTo
annmapDetails
annmapAll
annmapRange
annmapFilters

## Examples

```
if(interactive()) {
  # Get the gene for 'tp53'
  gene        = symbolToGene( 'tp53' )
  # And the transcripts for this gene
  transcripts = geneToTranscript( symbolToGene( 'tp53' ) )
  # And the proteins for this transcript
  proteins    = transcriptToProtein( transcripts )

 # get the transcript coords for the transcripts of this gene, at the start of this gene
  genomeToTranscriptCoords( start( gene ), transcripts, as.vector=TRUE )
  #Returns a vector:
  # ENST00000413465 ENST00000359597 ENST00000504290 ENST00000510385 ENST00000504937
  #            1018              NA              NA              NA              NA
  # ENST00000269305 ENST00000455263 ENST00000420246 ENST00000445888 ENST00000396473
  #              NA              NA              NA              NA              NA
  # ENST00000545858 ENST00000419024 ENST00000509690 ENST00000514944 ENST00000505014
  #              NA              NA              NA              NA              NA
  # ENST00000414315 ENST00000508793 ENST00000503591
  #              NA              NA              NA

  # With as.vector=FALSE
  genomeToTranscriptCoords( start( gene ), transcripts )
  # RangedData with 1 row and 1 value column across 1 space
  #             space       ranges | coord.space
  #       <character>    <IRanges> | <character>
  # 1 ENST00000413465 [1018, 1018] |  transcript

  genomeToProteinCoords( start( gene ), proteins, as.vector=TRUE )
  # ENSP00000410739 ENSP00000352610 ENSP00000269305 ENSP00000398846 ENSP00000391127
  #             340              NA              NA              NA              NA
  # ENSP00000391478 ENSP00000379735 ENSP00000437792 ENSP00000402130 ENSP00000425104
  #              NA              NA              NA              NA              NA
  # ENSP00000423862 ENSP00000394195 ENSP00000424104 ENSP00000426252
  #              NA              NA              NA              NA

  # With as.vector=FALSE
  genomeToProteinCoords( start( gene ), proteins )
  # RangedData with 1 row and 2 value columns across 1 space
  #             space     ranges |     frame coord.space
  #       <character>  <IRanges> | <numeric> <character>
  # 1 ENSP00000410739 [340, 340] |         0     protein
}
```

annmapDetails *annmap 'details' functions*

## Description

Get detailed annotations for the specified features.

## Usage

```
arrayDetails( ids, as.data.frame=FALSE )
chromosomeDetails( ids, as.data.frame=FALSE )
domainDetails( ids, as.data.frame=FALSE )
estExonDetails( ids, as.data.frame=FALSE )
estGeneDetails( ids, as.data.frame=FALSE )
estTranscriptDetails( ids, as.data.frame=FALSE )
exonDetails( ids, as.data.frame=FALSE )
geneDetails( ids, as.data.frame=FALSE )
predictionTranscriptDetails( ids, as.data.frame=FALSE )
probeDetails( ids, as.data.frame=FALSE )
probesetDetails( ids, as.data.frame=FALSE )
proteinDetails( ids, as.data.frame=FALSE )
synonymDetails( ids, as.data.frame=FALSE )
transcriptDetails( ids, as.data.frame=FALSE )
```

## Arguments

| | |
|---|---|
| ids | Database identifiers for the features of interest |
| as.data.frame | If FALSE, data will be converted to a GRanges object if possible, otherwise a data.frame |

## Value

Results in an GRanges object (or a data.frame if TRUE is passed for the second parameter), one \'row\' per feature, containing detailed annotations.

## Author(s)

Tim Yates

## See Also

annmapTo
annmapAll
annmapRange
annmapUtils
annmapFilters
GRanges

## Examples

```
if(interactive()) {
  annmapConnect()
  geneDetails(symbolToGene("TP53"))
}
```

---

annmapEnv *annmap 'env' functions*

---

## Description

Functions to access internal parameters

## Usage

```
annmapEnv()
annmapGetParam( key )
annmapSetParam( ... )
```

## Arguments

| | |
|---|---|
| ... | A list of key-value parameters you wish to set. |
| key | The key for the value you want to return. |

## Details

These functions allow some access to annmap\'s configuration data. They are included to help debug database connection issues, and are not normally needed.

On connection, a default arraytype (Affymetrix Exon arrays, where available) is specfied for the probe mappings. arrayType allows a different type of array to be specfied. This included for future compatibility.

## Author(s)

Tim Yates Crispin J. Miller

## See Also

annmapTo
annmapDetails
annmapAll
annmapRange
annmapFilters

## Examples

```
if(interactive()) {
  annmapEnv()
  annmapGetParam( "debug" )
  annmapConnect()
  annmapSetParam( debug=TRUE)
  annmapConnect()
  annmapSetParam( debug=FALSE)
  annmapDisconnect()
}
```

---

annmapFilters *annmap 'filter' functions*

---

## Description

Functions to filter exon array probeset names by the genome features they correspond to.

## Usage

```
exonic( probesets, exclude=FALSE )
hasProbes( probesets, num.probes=4, exclude=FALSE )
hasProbesAtleast( probesets, num.probes=4, exclude=FALSE )
hasProbesIn( probesets, num.probes=c( 1, 2, 3, 4 ), exclude=FALSE )
hasProbesBetween( probesets, min.probes=1, max.probes=4, exclude=FALSE, inclusive=TRUE )
intergenic( probesets, exclude=FALSE )
intronic( probesets, exclude=FALSE )
isExonic( probesets )
isIntergenic( probesets )
isIntronic( probesets )
isUnreliable( probesets )
unreliable( probesets, exclude=FALSE )
```

## Arguments

| | |
|---|---|
| probesets | A vector of probesets to filter |
| num.probes | The required number of probes to have in the probeset |
| exclude | If FALSE, then return a list containing only those probesets matching the filter. If TRUE then return only those that don\'t match the filter |
| min.probes | Minimum number of probes within a probeset |
| max.probes | Maximum number of probes within a probeset |
| inclusive | Whether to include the extremes of the range in the search or not |

## Details

Probesets are classified according to whether they map to known genes. The function exonic filters for probesets for which all probes match once (and only once) to the genome, and every probe hits an exon. Note that this means that a probeset that hits more than one exon, will be flagged as exonic. All probes in intronic probesets hit the genome once (and once only), and all probes hit a gene - however one or more probes hit an intron. intergenic probesets hit the genome once (and

once only) but one or more probes miss a gene compeletely. `unreliable` probesets comprise those that have at least one probe that does not align to the genome, or one or more probes that align at multiple loci (multiply targeted).

The functions `is.exonic`, `is.intronic` and `is.intergenic`, return a logical vector classifying the supplied probesets.

The functions `has.probes`, `has.probes.in` and `has.probes.between` can be used to filter a set of probesets according to the numbers of probes they contain.

## Author(s)

Tim Yates Crispin J. Miller

## See Also

`annmapTo`
`annmapDetails`
`annmapAll`
`annmapRange`
`annmapFilters`

## Examples

```
if(interactive()){
  annmapConnect()
  ps <- geneToProbeset(symbolToGene("TP53"))
  exonic(ps)
  intronic(ps)
  intergenic(ps)
  unreliable(ps)
  isExonic(ps)
  isIntronic(ps)
  isIntergenic(ps)
  isUnreliable(ps)
  hasProbes(ps)
  hasProbesIn(ps,1:3)
  hasProbesBetween(ps,2,3)
  hasProbesAtleast(ps,4)
}
```

---

# annmapRange *annmap 'range' functions*

---

## Description

Get the features within the specified genome coordinates.

## Usage

```
  domainInRange( x, ..., as.vector = FALSE )
  ## S4 method for signature 'GRanges'
domainInRange( x, as.vector=FALSE )
  ## S4 method for signature 'RangedData'
domainInRange( x, as.vector=FALSE )
  ## S4 method for signature 'character'
domainInRange( x, start, end, strand, ..., as.vector=FALSE )
  ## S4 method for signature 'data.frame'
domainInRange( x, as.vector=FALSE )
  ## S4 method for signature 'NULL'
domainInRange( x, as.vector=FALSE )
  ## S4 method for signature 'factor'
domainInRange( x, start, end, strand, ..., as.vector=FALSE )

  estExonInRange( x, ..., as.vector = FALSE )
  ## S4 method for signature 'GRanges'
estExonInRange( x, as.vector=FALSE )
  ## S4 method for signature 'RangedData'
estExonInRange( x, as.vector=FALSE )
  ## S4 method for signature 'character'
estExonInRange( x, start, end, strand, ..., as.vector=FALSE )
  ## S4 method for signature 'data.frame'
estExonInRange( x, as.vector=FALSE )
  ## S4 method for signature 'NULL'
estExonInRange( x, as.vector=FALSE )
  ## S4 method for signature 'factor'
estExonInRange( x, start, end, strand, ..., as.vector=FALSE )

  estGeneInRange( x, ..., as.vector = FALSE )
  ## S4 method for signature 'GRanges'
estGeneInRange( x, as.vector=FALSE )
  ## S4 method for signature 'RangedData'
estGeneInRange( x, as.vector=FALSE )
  ## S4 method for signature 'character'
estGeneInRange( x, start, end, strand, ..., as.vector=FALSE )
  ## S4 method for signature 'data.frame'
estGeneInRange( x, as.vector=FALSE )
  ## S4 method for signature 'NULL'
estGeneInRange( x, as.vector=FALSE )
  ## S4 method for signature 'factor'
estGeneInRange( x, start, end, strand, ..., as.vector=FALSE )

  estTranscriptInRange( x, ..., as.vector = FALSE )
  ## S4 method for signature 'GRanges'
estTranscriptInRange( x, as.vector=FALSE )
  ## S4 method for signature 'RangedData'
estTranscriptInRange( x, as.vector=FALSE )
  ## S4 method for signature 'character'
estTranscriptInRange( x, start, end, strand, ..., as.vector=FALSE )
  ## S4 method for signature 'data.frame'
estTranscriptInRange( x, as.vector=FALSE )
```

```
  ## S4 method for signature 'NULL'
estTranscriptInRange( x, as.vector=FALSE )
  ## S4 method for signature 'factor'
estTranscriptInRange( x, start, end, strand, ..., as.vector=FALSE )

  exonInRange( x, ..., as.vector = FALSE )
  ## S4 method for signature 'GRanges'
exonInRange( x, as.vector=FALSE )
  ## S4 method for signature 'RangedData'
exonInRange( x, as.vector=FALSE )
  ## S4 method for signature 'character'
exonInRange( x, start, end, strand, ..., as.vector=FALSE )
  ## S4 method for signature 'data.frame'
exonInRange( x, as.vector=FALSE )
  ## S4 method for signature 'NULL'
exonInRange( x, as.vector=FALSE )
  ## S4 method for signature 'factor'
exonInRange( x, start, end, strand, ..., as.vector=FALSE )

  geneInRange( x, ..., as.vector = FALSE )
  ## S4 method for signature 'GRanges'
geneInRange( x, as.vector=FALSE )
  ## S4 method for signature 'RangedData'
geneInRange( x, as.vector=FALSE )
  ## S4 method for signature 'character'
geneInRange( x, start, end, strand, ..., as.vector=FALSE )
  ## S4 method for signature 'data.frame'
geneInRange( x, as.vector=FALSE )
  ## S4 method for signature 'NULL'
geneInRange( x, as.vector=FALSE )
  ## S4 method for signature 'factor'
geneInRange( x, start, end, strand, ..., as.vector=FALSE )

  predictionTranscriptInRange( x, ..., as.vector = FALSE )
  ## S4 method for signature 'GRanges'
predictionTranscriptInRange( x, as.vector=FALSE )
  ## S4 method for signature 'RangedData'
predictionTranscriptInRange( x, as.vector=FALSE )
  ## S4 method for signature 'character'
predictionTranscriptInRange( x, start, end, strand, ..., as.vector=FALSE )
  ## S4 method for signature 'data.frame'
predictionTranscriptInRange( x, as.vector=FALSE )
  ## S4 method for signature 'NULL'
predictionTranscriptInRange( x, as.vector=FALSE )
  ## S4 method for signature 'factor'
predictionTranscriptInRange( x, start, end, strand, ..., as.vector=FALSE )

  probesetInRange( x, ..., as.vector = FALSE )
  ## S4 method for signature 'GRanges'
probesetInRange( x, as.vector=FALSE )
  ## S4 method for signature 'RangedData'
probesetInRange( x, as.vector=FALSE )
```

```
  ## S4 method for signature 'character'
probesetInRange( x, start, end, strand, ..., as.vector=FALSE )
  ## S4 method for signature 'data.frame'
probesetInRange( x, as.vector=FALSE )
  ## S4 method for signature 'NULL'
probesetInRange( x, as.vector=FALSE )
  ## S4 method for signature 'factor'
probesetInRange( x, start, end, strand, ..., as.vector=FALSE )

  probeInRange( x, ..., as.vector = FALSE )
  ## S4 method for signature 'GRanges'
probeInRange( x, as.vector=FALSE )
  ## S4 method for signature 'RangedData'
probeInRange( x, as.vector=FALSE )
  ## S4 method for signature 'character'
probeInRange( x, start, end, strand, ..., as.vector=FALSE )
  ## S4 method for signature 'data.frame'
probeInRange( x, as.vector=FALSE )
  ## S4 method for signature 'NULL'
probeInRange( x, as.vector=FALSE )
  ## S4 method for signature 'factor'
probeInRange( x, start, end, strand, ..., as.vector=FALSE )

  proteinInRange( x, ..., as.vector = FALSE )
  ## S4 method for signature 'GRanges'
proteinInRange( x, as.vector=FALSE )
  ## S4 method for signature 'RangedData'
proteinInRange( x, as.vector=FALSE )
  ## S4 method for signature 'character'
proteinInRange( x, start, end, strand, ..., as.vector=FALSE )
  ## S4 method for signature 'data.frame'
proteinInRange( x, as.vector=FALSE )
  ## S4 method for signature 'NULL'
proteinInRange( x, as.vector=FALSE )
  ## S4 method for signature 'factor'
proteinInRange( x, start, end, strand, ..., as.vector=FALSE )

  transcriptInRange( x, ..., as.vector = FALSE )
  ## S4 method for signature 'GRanges'
transcriptInRange( x, as.vector=FALSE )
  ## S4 method for signature 'RangedData'
transcriptInRange( x, as.vector=FALSE )
  ## S4 method for signature 'character'
transcriptInRange( x, start, end, strand, ..., as.vector=FALSE )
  ## S4 method for signature 'data.frame'
transcriptInRange( x, as.vector=FALSE )
  ## S4 method for signature 'NULL'
transcriptInRange( x, as.vector=FALSE )
  ## S4 method for signature 'factor'
transcriptInRange( x, start, end, strand, ..., as.vector=FALSE )
```

## Arguments

| | |
|---|---|
| as.vector | If TRUE returns a vector of database identifiers. If FALSE returns a GRanges object containing detailed annotation. |
| x | The name of the chromosome of interest – in the case of the factor or character variants), or a GRanges object or data.frame containing location information. In the case of a data.frame, columns must be named chr or chromosome_name, followed by start, end and strand. RangedData objects must contain a strand in their meta-data. And strand must be 1 or -1 in all cases arart from GRanges where it obviously has to be + or -. All of the NULL variants simply return NULL, in-keeping with the fluent style of the rest of the package. |
| start | Start of the region |
| end | End of the region |
| strand | 1 == top stand, -1 == bottom strand |
| ... | The ellipsis is to allow this multi-method style of programming. |

## Details

Find all the specified features within a given region of the genome. For all functions except probeInRange, features that fall on the boundaries of the region (i.e. are partially overlapping) are returned too. For probeInRange probes that span the start of the range are NOT returned (but those spanning the end of the range are).

The function annmapRangeApply makes it possible to map any of these functions down the rows of a RangedData or GRanges object. The defaults are set up so that it will handle the output of one of the InRange methods here. This makes it easy to nest functions, for example, to find all genes in a given region of the the genome, and then find the exon array probes that map to those genes (see below).

## Value

Returns a GRanges object, one \'row\' per feature, containing detailed annotations, or a vector of identifiers, depending on the value of as.vector.

## Author(s)

Tim Yates

## See Also

annmapTo
annmapDetails
annmapAll
annmapUtils
annmapFilters
RangedData GRanges

## Examples

```
if(interactive()) {
  annmapConnect()

  r = geneInRange( '17', 7510000, 7550000, 1 )
```

```
  # Can take equal length vectors as parameters
 geneInRange( c( '17', 'X' ), c( 7510000, 1000000 ), c( 7550000, 1500000 ), c( -1, -1 ) )

  # Or a data.frame
df = data.frame( chr=c( '17', 'X' ), start=c( 7510000, 1000000 ), end=c( 7550000, 1500000 ), strand=c( -1, -1
  geneInRange( df )

  # Or RangedData objects
  transcriptInRange( geneDetails( symbolToGene( c( 'tp53', 'ssh' ) ) ) )
}
```

---

annmapSeqname *Seqnames manipulation functions*

---

## Description

These functions allow easier manipulation of the seqnames column of a GRanges object

## Usage

```
generalisedNameToNCBI( name, ... )
generalisedNameToEnsembl( name, ... )
seqnameMapping( x, mappingFunction, ... )
seqnamesToNCBI( x )
seqnamesToEnsembl( x )
```

## Arguments

| | |
|---|---|
| name | The name to convert. |
| x | A GRanges object to convert the seqnames of. |
| mappingFunction | The function to do the mapping of names. |
| ... | Other arguments you may wish to send to a custom mapping function. |

## Details

These functions allow simple mapping between seqnames of a GRanges object.

The two standard derivations are `seqnamesToNCBI` and `seqnamesToEnsembl`. The rules for mapping are:

```
Ensembl      NCBI
1       <=>  chr1
...
22      <=>  chr22
X       <=>  chrX
Y       <=>  chrY
MT      <=>  chrM
```

You can define your own mapping function and pass it as the `mappingFunction` parameter to `seqnameMapping` function to do your own custom mapping.

The function `seqnamesToNCBI` calls `seqnameMapping` with `generalisedNameToNCBI` as the `mappingFunction`. The function `seqnamesToEnsembl` uses `generalisedNameToEnsembl`.

### Author(s)

Tim Yates

### Examples

```
if(interactive()) {
  annmapConnect()
  seqnamesToNCBI( symbolToGene( c( 'tp53', 'shh' ) ) )
}
```

---

annmapTo *annmap 'to' functions*

---

### Description

Map between the different levels of annotation in Annmap. For example, given a vector of gene identifiers, geneToExon will return the exons in those genes.

### Usage

```
arrayToProbeset( ids, as.vector=FALSE )
domainToGene( ids, as.vector=FALSE )
domainToProbeset( ids, as.vector=FALSE )
domainToProtein( ids, as.vector=FALSE )
domainToTranscript( ids, as.vector=FALSE )
estExonToEstGene( ids, as.vector=FALSE )
estExonToEstTranscript( ids, as.vector=FALSE )
estExonToProbeset( ids, as.vector=FALSE )
estGeneToEstExon( ids, as.vector=FALSE )
estGeneToEstTranscript( ids, as.vector=FALSE )
estGeneToProbeset( ids, as.vector=FALSE )
estTranscriptToEstExon( ids, as.vector=FALSE )
estTranscriptToEstGene( ids, as.vector=FALSE )
estTranscriptToProbeset( ids, as.vector=FALSE )
exonToGene( ids, as.vector=FALSE )
exonToProbeset( ids, as.vector=FALSE )
exonToTranscript( ids, as.vector=FALSE )
geneToDomain( ids, as.vector=FALSE )
geneToExon( ids, as.vector=FALSE )
geneToExonProbeset( ids, as.vector=FALSE, probes.min=4 )
geneToExonProbesetExpr( x, ids, probes.min=4 )
geneToProbeset( ids, as.vector=FALSE )
geneToProtein( ids, as.vector=FALSE )
geneToSymbol( ids )
geneToSynonym( ids, as.vector=FALSE )
geneToTranscript( ids, as.vector=FALSE )
predictionTranscriptToPredictionExon( ids )
predictionTranscriptToProbeset( ids, as.vector=FALSE )
probeToHit( ids, as.data.frame=FALSE )
probeToProbeset( ids, as.vector=FALSE )
probesetToCdnatranscript( ids, as.vector=FALSE, rm.unreliable=TRUE )
```

```
probesetToDomain( ids, as.vector=FALSE, rm.unreliable=TRUE )
probesetToEstExon( ids, as.vector=FALSE, rm.unreliable=TRUE )
probesetToEstGene( ids, as.vector=FALSE, rm.unreliable=TRUE )
probesetToEstTranscript( ids, as.vector=FALSE, rm.unreliable=TRUE )
probesetToExon( ids, as.vector=FALSE, rm.unreliable=TRUE )
probesetToGene( ids, as.vector=FALSE, rm.unreliable=TRUE )
probesetToHit( ids, as.data.frame=FALSE, rm.unreliable=TRUE )
probesetToPredictionTranscript( ids, as.vector=FALSE, rm.unreliable=TRUE )
probesetToProbe( ids, as.vector=FALSE )
probesetToProtein( ids, as.vector=FALSE, rm.unreliable=TRUE )
probesetToTranscript( ids, as.vector=FALSE, rm.unreliable=TRUE )
proteinToDomain( ids, as.vector=FALSE )
proteinToGene( ids, as.vector=FALSE )
proteinToProbeset( ids, as.vector=FALSE )
proteinToTranscript( ids, as.vector=FALSE )
symbolToEstGene( ids, as.vector=FALSE )
symbolToEstTranscript( ids, as.vector=FALSE )
symbolToGene( ids, as.vector=FALSE )
symbolToTranscript( ids, as.vector=FALSE )
synonymToEstGene( ids, as.vector=FALSE )
synonymToEstTranscript( ids, as.vector=FALSE )
synonymToGene( ids, as.vector=FALSE )
synonymToTranscript( ids, as.vector=FALSE )
transcriptToCdnaprobeset( ids, as.vector=FALSE )
transcriptToDomain( ids, as.vector=FALSE )
transcriptToExon( ids, as.vector=FALSE )
transcriptToExonProbeset( ids, as.vector=FALSE, probes.min=4 )
transcriptToGene( ids, as.vector=FALSE )
transcriptToProbeset( ids, as.vector=FALSE )
transcriptToProtein( ids, as.vector=FALSE )
transcriptToSynonym( ids, as.vector=FALSE )
transcriptToTranslatedprobes( ids )
```

## Arguments

| | |
|---|---|
| as.vector | If TRUE returns a vector of database identifiers. If FALSE returns a link{RangedData} object containing detailed annotation. |
| as.data.frame | Where a vector is inappropriate for the data type, the option to return the data as a plain data.frame in place of a GRanges object is given. |
| ids | Database identifiers to map from. Can be either a vector of database identifiers, or a GRanges object. |
| probes.min | How many probes need to match before the probeset is returned. |
| rm.unreliable | If TRUE, the input probeset list is filtered, and all unreliable probesets are removed. |
| x | An ExpressionSet object or a matrix containing expression data. If the latter, then the rownames must specify the exon array probeset names. |

## Details

In most cases, these functions should be self-explantory. However, by default, the mappings involving probes and probesets do some filtering of the data. This means that probesets which have one

or more probes that don't match to the genome, or which match to multiple loci, are removed (see unreliable for more details).

The function transcriptToTranslatedprobes returns a list of GRanges objects (one for each transcript) containing each probe that hits that translated transcripts and the relative start and end locations.

## Value

Results in an GRanges object, one row per feature, containing detailed annotations, or a vector, as defined by as.vector.

## Author(s)

Tim Yates

## See Also

annmapDetails
annmapAll
annmapRange
annmapUtils
annmapFilters
link{GRanges}

## Examples

```
if(interactive()) {
  annmapConnect()
  geneToExon(symbolToGene("TP53"))
}
```

---

annmapUtils *annmap 'utils' functions*

---

## Description

Functions to connect to the database and manage the database connections.

## Usage

```
annmapConnect( name, use.webservice=FALSE, quiet.webservice=FALSE )
annmapDisconnect()
annmapAddConnection( dsname, species, version,
                     host='localhost',
                     username=as.character( Sys.info()[ 'user' ] ),
                     password='',
                     port='',
                     overwrite=FALSE,
                     testConnect=TRUE )
arrayType( name=NULL, pick.default=FALSE, silent=FALSE )
annmapToggleCaching()
annmapClearCache()
```

```
annmapRangeApply( x, f, filter=c( chr="space", start="start", end="end", strand="strand" ), coerce
 strandAsInteger( granges )
 geneToGeneRegionTrack( genes, genome, coalesce.name=NULL, ... )
```

## Arguments

| | |
|---|---|
| name | The name of the database to connect to, or the array to select. |
| use.webservice | If TRUE, we will use the annmap webservices rather than a local MySQL installation. |
| quiet.webservice | If FALSE, there will be output as the webservice calls are processed. Set TRUE to silence these. |
| dsname | The name of the datasource to add or modify. |
| species | The species of interest. |
| version | The version of the database to connect to. |
| host | The location of the MySQL installation. |
| username | The username to connect to MySQL. |
| password | The password required to connect to MySQL. |
| port | The port MySQL is running on. (Use NA for default) |
| overwrite | If another connection with this dsname already exists, should it be overwritten? |
| testConnect | If TRUE, the connection will be attempted before adding it to the databases.txt file. |
| pick.default | If TRUE, arrayType will choose the first available arraytype for this species. |
| silent | If TRUE, it will skip telling you which array you have selected. |
| x | A RangedData object |
| f | A function to apply to each \'row\' of the RangedData object |
| filter | Which \'columns\' of the RangedData object does the function need, and what parameters in the function do they map on to?. For example, by default, the field \'space\' gets mapped to the parameter \'chr\'. |
| coerce | What is the type of each parameter in \'f\'? |
| ... | additional parameters |
| granges | A GRanges object |
| genes | The genes you wish to load into a GeneRegionTrack they must all be on the same chromosome. |
| genome | A valid Gviz genome, ie: 'hg19'. |
| coalesce.name | If this is a character vector, all genes will be joined into a single track with this name. Otherwise each gene will have its own track. |

## Details

annmapConnect is used to establish a connection to an instance of the Annmap database, and annmapDisconnect closes the connection.

arrayType is used to specify the array you wish to use for queries based on Affymetrix probesets.

Many of the functions in annmap cache results locally. The function annmapToggleCaching turns this functionality on and off, and annmapClearCache can be used to clear the cache (this is not normally something a user needs to do).

Note that details of how to set up the default databases, connection details, etc. Can be found in the package vignette.

The function `strandAsInteger` takes a `GRanges` object and returns an integer vector of strands in the Ensembl style. "+" becomes 1, "-" becomes -1, and "*" becomes `NA`.

The function `geneToGeneRegionTrack` takes a list of genes (character vector, GRanges object, etc), and returns a list of GeneRegionTracks which can be plotted in Gviz. There is an example in the cookbook.

## Author(s)

Tim Yates Crispin J. Miller

## See Also

`annmapTo`
`annmapDetails`
`annmapAll`
`annmapRange`
`annmapFilters`

## Examples

```
if(interactive()) {
  annmapConnect()
  annmapToggleCaching()
  annmapToggleCaching()

  annmapRangeApply(symbolToGene("TP53",as.vector=FALSE),probeInRange)

  #NOTE: since the next function empties out the local cache, don't
  #run it unless you want to do this!
  #annmapClearCache()
}
```

---

`annmapUtr` *annmap coding functions*

---

## Description

Functions to deal with coding regions and UTRs

## Usage

```
transcriptToUtrRange( ids, end=c( "both", "5", "3" ), as.data.frame=FALSE, on.translation.error=s
transcriptToUtrExon( ids, end=c( 'both', '5', '3' ), as.vector=FALSE, on.translation.error=stop )
transcriptToCodingRange( ids, end=c( "both", "5", "3" ), as.data.frame=FALSE, on.translation.erro
transcriptToCodingExon( ids, end=c( 'both', '5', '3' ), as.vector=FALSE, on.translation.error=sto
utrProbesets( probesets, transcripts, end=c( "both", "5", "3" ), on.translation.error=stop )
codingProbesets( probesets, transcripts, end=c( "both", "5", "3" ), on.translation.error=stop )
nonIntronicTranscriptLength( ids, end=c( 'none', 'both', '5', '3' ), on.translation.error=stop )
 nonIntronicGeneLength( ids )
```

## Arguments

| | |
|---|---|
| ids | A vector of Transcript Names, or a RangedData object of Transcripts returned from another annmap call. |
| as.data.frame | If FALSE, data will be converted to a RangedData object if possible, otherwise a data.frame |
| as.vector | If TRUE returns a vector of database identifiers. If FALSE returns a link{GRanges} object containing detailed annotation. |
| probesets | An optional vector of Probeset Names, or a RangedData object of Probesets returned from another annmap call. |
| transcripts | An optional vector of Transcript Names, or a RangedData object of Transcripts returned from another annmap call. |
| end | Which end ("both", "3" or "5") of the Transcript(s) you are interested in (defaults to both). |
| on.translation.error | A function to call with a character vector explaining the problem if one is encountered with the translation locations in the database. |

## Details

The first two functions given here, transcriptToUtrRange and transcriptToCodingRange return the transcripts of interest, with their ranges adjusted depending on the UTR of each.

With transcriptToUtrRange, a RangedData object is returned with the name of the transcript, the end in question, and the genomic location of that UTR. If both is passed as the end parameter, then each transcript will generate up to two rows in the returned object. It may return less than two rows if the end parameter is used, or if there is no UTR for the end specified. (A Transcript with no UTR will return zero results)

The transcriptToCodingRange function returns the same as calling transcriptDetails, but with the start and end locations modified by the range of the UTR. If end is passed, then only the UTR at this end will be taken into consideration and used to modify the returned location.

The transcriptToCodingExon and transcriptToUtrExon functions return the exons for each transcript limited to only those exons (or partions thereof) which are coding or part of the UTR.

utrProbesets and codingProbesets are functions to find or filter probesets which have probes targeting the type of region specified by the function name.

A call to utrProbesets with a list of Probesets will return those probesets that have at least one probe hitting the UTR of any transcript.

A call to utrProbesets with a list of Probesets and a list of Transcripts will return those probesets the have at least one probe hitting the UTR of any of the specified Transcripts.

A call to utrProbesets with only the probesets parameter omitted, will return all probesets which have at least one probe in the UTR region of the specified Transcripts.

You cannot omit both the Probesets and Transcripts parameters simultaneously.

The codingProbesets method does the inverse of the utrProbesets function: it returns probesets having at least one probe in the coding region of a Transcipt (or the specified Transcripts).

Note that the UTR of a Transcript includes the intronic UTR regions, and the coding region of a Transcript includes the intronic coding regions.

This means that utrProbesets and codingProbesets can sometimes return intronic and/or intergenic probesets. These can be removed with a call to the appropriate filter function (see examples).

All unreliable probesets are automatically removed by these functions before mapping.

Calling nonIntronicTranscriptLength will return the length of the exons (coding can be specified via the end parameter) in a given list of transcripts.

And nonIntronicGeneLength will give the length of all exons in a given gene when overlaps are taken into account (so two exactly overlapping exons will count once for the length)

## Author(s)

Tim Yates

## See Also

annmapTo
annmapDetails
annmapAll
annmapRange
annmapFilters

## Examples

```
if(interactive()) {
  # Only return exonic probesets hitting the UTRs of ENST00000414566
  exonic( utrProbesets( NULL, "ENST00000414566" ) )
}
```

---

# genomicPlotting *Plotting a section of a chromosome.*

---

## Description

These functions are used when we need to plot one or both strands of a section of chromosome.

## Usage

```
genomicPlot( xrange, gene.area.height=NULL, gene.layout.padding=100, highlights=NULL, draw.oppos
         padding.lines=1, .genes=NULL, .exons=NULL, invert.strands=FALSE, draw.scale=TRUE, ... )
genomicExonDepthPlot( .exons, start, end, exon.depth.alpha=0.1, exon.depth.col='black', ... )
genomicProbePlot( probes, start, end, probe.col='green', probe.alpha=0.3, ... )
```

## Arguments

| | |
|---|---|
| xrange | An IRanges object representing the region of interest (with a strand if reqd) |
| gene.area.height | If NULL then both strands to max height of either of them, else if NA then both strands limited to their implied height otherwise, if an integer then both strands limited to the specified height |
| gene.layout.padding | How much space (in bases) needs to be between each gene in a layer. Needed to stop gene names overlapping |

| | |
|---|---|
| highlights | You can pass this a data.frame of values to render as dummy genes in the view. Columns MUST include start, end, strand and name. It may also optionally include the columns col to specify a per-gene background colour, or bor to specify the colour to be used for the gene border and the label text. If these two are not passed, sensible defaults are chosen automatically. |
| draw.opposite.strand | Do we draw a washed out representation of the other strand. Only applies if strand( xrange ) != '*' |
| exon.depth.plot | Should we draw the exondepth? set to NULL if not |
| padding.lines | How much padding above and below the plot (in grid lines) |
| .genes | Optionally pass in the pre-loaded genes and exons (then we skip loading them in this function) |
| .exons | The exons that are to be used |
| invert.strands | Should the forward strand be on the bottom of the plot? |
| draw.scale | Draw a scale between the two strands? |
| ... | Parameters passed on to functions called by this function |
| exon.depth.alpha | The transparency for the exon.depth rectangles |
| exon.depth.col | The color for the exon.depth rectangles |
| start | The start of the region of interest |
| end | The end of the region of interest |
| probe.alpha | How transparent should probes be rendered? |
| probe.col | The colour to use for probes. |
| probes | The probes for the region of interest (as a data.frame). |

## Author(s)

Tim Yates

---

ngsPlot *Plotting BAM file data alongside the features of a chromosome*

---

## Description

These functions aid plotting a-la xmapbridge but in a format that is more publication friendly

## Usage

```
 # Utility Methods
convertBamToRle( bam.file.name, chr, start, end, chr.name.mapping=function( name ){ name } )
 generateBridgeData( xrange, bamFiles, colours=NULL, names=NULL )
 ngsTraceScale( vector.of.xbams.and.ybams )
 ngsTraceLabel( rle.data )
ngsTracePlotter( rle.data, start, end, ylim, trace.label.properties=list(), smoothing.function=fu
            trace.clip='inherit', trace.draw.scale=FALSE, trace.bor='transparent', trace.pad=c(
```

```
 # Plotting Methods
ngsBridgePlot( xrange, data=list(), main=NULL, sub=NULL, highlights=NULL, trace.plotter=ngsTraceP
         trace.scale=ngsTraceScale, trace.draw.scale=NULL, trace.match.strand=TRUE, probe.plot=N
              .genes=NULL, .exons=NULL, ... )
```

## Arguments

| | |
|---|---|
| bam.file.name | The name of the BAM file to read in |
| chr | The chromosome of interest. |
| start | The start of the region of interest |
| end | The end of the region of interest |
| chr.name.mapping | The function to convert between the Annmap chr name to the chr name in the BAM file. By default, this just uses chr supplied as the parameter, however it can be set to any function you like. One example of this is generalisedNameToNCBI |
| xrange | The genomic range for the x-axis. Should be a GRanges object. |
| bamFiles | A vector containing the filenames of your BAM files. |
| colours | A vector of colours for each file (sensible defaults will be chosen if NULL). |
| names | A vector of names to show on the traces drawn by ngsTracePlotter |
| vector.of.xbams.and.ybams | The trace.scale function is passes a vector of the elements of xbams and ybams concatenated together. |
| rle.data | A list containing fields rle (the Rle data to be plotted), name (the name of the Trace) and col (the colour for the trace). |
| ylim | A vector of min and max values for this plot (usually retrieved from ngs.trace.scale) |
| trace.label.properties | Properties to be sent to the grid.text call for plotting the label on the trace. To hide the label, this should be NA. |
| smoothing.function | A function that generates a smoothed RLE object. |
| trace.clip | Is the trace clipped to it's bounding box? One of 'inherit', 'on' or 'off'. See viewport. |
| trace.draw.scale | If TRUE, x and y scales are drawn with main=TRUE (see grid.xaxis and grid.yaxis), if FALSE, then neither axis is drawn. You can control individual axis drawing by passing a vector such as trace.draw.scale='x' to just draw the x axis. You can also pass a list such as trace.draw.scale=list(x=TRUE,y=FALSE), and this will draw both the x and y axis, but pass main=TRUE to the grid.xaxis call, and main=FALSE to grid.yaxis |
| trace.bor | The colour for a box that is drawn round this trace.plot. |
| trace.pad | A 2 element vector consisting of the number of 'lines' of padding to allow at the top and bottom of the plot respecively |
| data | A list containing an element per trace. Each element of this list is, in turn, passed to the trace.plotter and trace.scale functions where the plotting happens – see details. ) |
| main | The main title for the plot. |
| sub | A sub-title for the plot. |

| | |
|---|---|
| highlights | Highlight regions for the plot. See genomicPlot. |
| trace.plotter | The function to call to draw the traces (see ngsTracePlotter) |
| genome.layout.weight | The weight for the genomic plot in the layout of this grid |
| trace.scale | Either a function to calculate the global max for the NGS traces (see ngsTraceScale) OR a 2 element vector cntaining the min and max extent of the trace. |
| trace.match.strand | If TRUE, we will only draw the rle data from the strand defined in xrange. If false, we will draw all of the rle data. Can also be set to '+' or '-' to only draw the trace from the given strand (ignoring the strand of xrange). |
| probe.plot | The function to plot the probes (see genomicProbePlot), NULL if not drawn. |
| exon.depth.plot | The function to draw the exon depth (see genomicExonDepthPlot), NULL if not drawn. |
| .genes | Optionally pass a list of genes to limit the plot to. |
| .exons | An optional list of exons to limit the plot to. |
| ... | Parameters passed on to functions called by this function |

## Details

convertBamToRle will take a BAM file name, and a region of interest and return a list() containing two elements, '+' and '-'. Each element will be an Rle object, one for each strand.

The data parameter to ngsBridgePlot is a list of elements as defined in the rle.data parameter, one element per NGS trace, ie:

```
    library(grid)
    library(annmap)

    # Connect to datasource with annmapConnect()

    # Ensure we have a clean plot
    grid.newpage()

    bamFiles = c( 'data1.bam', 'data2.bam', 'data3.bam' )
    colours  = rainbow( 3, v=0.5, s=0.5 )
    data = lapply( seq_along( bamFiles ), function( idx ) {
      list( rle=convertBamToRle( bamFiles[ idx ], 'I', 40000, 100000 ),
            col=colours[ idx ],
            name=paste( 'Trace', bamFiles[ idx ] ) )
    } )
  ngsBridgePlot( RangedData( space='I', ranges=IRanges( 40000, 100000 ) ), data=data, main='Exampl
```

## Author(s)

Tim Yates

## See Also

genomicProbePlot, genomicPlot, genomicExonDepthPlot

spliceIndex *Splice indexing*

## Description

Calculates the splicing index for the probesets in one or more genes, as defined in the Affymetrix white paper "Alternative Transcript Analysis Methods for Exon Arrays".

## Usage

```
 spliceGroupIndex( x, group.column, members )
spliceIndex( x, ids, group, gps, group.index.fn=spliceGroupIndex, median.gene=FALSE, median.probe
```

## Arguments

| | |
|---|---|
| x | eSet containing expression data |
| group.column | a column name for the group data |
| members | a set of arrays |
| ids | Character vector of Ensembl gene names |
| group | If defined, the column name in the ExpressionSet's pData object in which to look for gps |
| gps | The two sets of arrays to compare |
| group.index.fn | a method which, when passed an ExpressionSet (from the Biobase package), a column name for the group data and a set of arrays, will return the indices of interest |
| median.gene | Use the median instead of the mean when calculating averages across genes |
| median.probeset | Use the median instead of the mean when calculating averages across probesets in each replicate group |
| unlogged | Unlog the expression data before calculating the splicing index (and then re-log afterwards) |

## Details

The splicing index gives a measure of the difference in expression level for each probeset in a gene between two sets of arrays, relative to the gene-level average in each set. This is calculated only for those probesets that are defined as exonic (See exonic).

The two sets of arrays can be specified in two ways: First, by using numeric indices defining the appropriate columns in the expression data. This is done by supplying these as a list to gps (e.g. gps=list(1:3,4:6) will calculate the splicing index between arrays 1,2,3 and 4,5,6. Alternatively, the annotation in the phenoData object from x can be used (e.g. group="treatment",gps=c("a","b") will compare between the arrays labelled 'a', and 'b' in the 'treatment' column of pData(x)).

The implementation also calculates a p.value and t.statistic for each probeset; these are returned alongside the splicing index.

By default, the splicing index is calculated using the mean across genes and samples. Specifying median.gene=TRUE or median.probeset=TRUE will use the median instead (for the gene or probeset level averages, respectively). It is calculated using the unlogged data, unless unlogged=FALSE. This only affects the internal calculations; values in x are always assumed to be logged, and the splicing index is always returned on the log2 scale.

## Author(s)

Tim Yates Crispin J. Miller

## See Also

exonic

## Examples

```
if(interactive()) {
  # Loads the Expression Set into x.rma
  load( '../unitTests/HuEx-1_0.tp53.expr.RData' )
  spliceIndex( x.rma, symbolToGene( 'tp53' ), gps=list(1:3,4:6) )
}
```

# Index