

Package ‘xmapcore’

October 8, 2013

Type Package

Title Core access to the xmap database (installed separately)

Version 1.15.0

Date 2010-01-11

Author Tim Yates <tyates@picr.man.ac.uk>

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

Depends R (>= 2.8.0), methods, IRanges

Imports DBI, RMySQL (>= 0.6-0), digest, Biobase

Suggests RUnit

Description xmapcore is deprecated and will not be supported after Bioconductor release 2.10. It has been superceded by a new package 'annmap'.
xmapcore allows mapping between genetic features and any available Affymetrix Exon arrays for Homo Sapiens, Mus Musculus, Rattus Norvegicus and Schizosaccharomyces Pombe.

License GPL-2

URL <http://xmap.picr.man.ac.uk>, <http://www.bioconductor.org>

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

Collate zzz.R db.R utils.R cache.R statements.R filtering.R utr.R plots.R coords.R

R topics documented:

xmapcore-package	2
xmapcore.all	3
xmapcore.coords	4
xmapcore.details	6
xmapcore.env	7

xmapcore.filters	8
xmapcore.range	10
xmapcore.to	11
xmapcore.utils	14
xmapcore.utr	15
Index	18

xmapcore-package	<i>Provide access to the X:Map annotation database</i>
------------------	--

Description

X:Map <http://xmap.picr.man.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>). X:Map also provides probe to genome mappings for Affymetrix Exon arrays.

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

xmapcore provides core annotation. Other packages, which depend on xmapcore, provide additional functionality:

xmapprotein: provides additional annotation for protein-coding genes.

xmapbridge) makes it possible to output graphs that can be displayed within the X:Map genome browser.

Details

Package: xmapcore
Type: Package
Version: 0.9.3
Date: 2009-07-08
License: GPL-2

Author(s)

Tim Yates
Crispin Miller
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

xmapprotein xmapbridge [IRanges](#)

xmapcore.all

xmapcore`all` functions

Description

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

Usage

```
all.arrays( as.vector=TRUE )
all.chromosomes( as.vector=TRUE )
all.domains( as.vector=TRUE )
all.est_exons( as.vector=TRUE )
all.est_genes( as.vector=TRUE )
all.est_transcripts( as.vector=TRUE )
all.exons( as.vector=TRUE )
all.genes( as.vector=TRUE )
all.prediction_transcripts( as.vector=TRUE )
all.probes( as.vector=TRUE )
all.probesets( as.vector=TRUE )
all.proteins( as.vector=TRUE )
all.symbols( as.vector=TRUE )
all.synonyms( as.vector=TRUE )
all.transcripts( as.vector=TRUE )
```

Arguments

`as.vector` If TRUE returns a vector of database identifiers. If FALSE returns a [RangedData](#) object containing detailed annotation.

Value

Returns a vector or [RangedData](#) object, as defined by `as.vector`.

Author(s)

Tim Yates

See Also

[xmapcore.to](#)
[xmapcore.details](#)
[xmapcore.range](#)
[xmapcore.utils](#)
[xmapcore.filters](#)
[RangedData](#)

Examples

```

if(interactive()) {
  xmap.connect()
  all.chromosomes()
  all.chromosomes(as.vector=FALSE)
}

```

xmapcore.coords	<i>xmapcore co-ordinate mapping functions</i>
-----------------	---

Description

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

Usage

```

transcript.coords.to.genome( transcript.ids, position=1, as.vector=TRUE, check.bounds=TRUE, truncate=TRUE )
genome.to.transcript.coords( position, transcript.ids, as.vector=TRUE, check.bounds=TRUE )
protein.coords.to.genome( protein.ids, position=1, as.vector=TRUE, check.bounds=TRUE, truncate=TRUE )
genome.to.protein.coords( position, protein.ids, as.vector=TRUE, check.bounds=TRUE )

```

Arguments

transcript.ids	A vector of transcript.ids (or a RangedData object of transcripts returned from another xmapcore function)
position	The position of interest (either a genomic position for both of the genome.to.xxx 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 position outside the range of the protein/transcript will cause a warning to be issued and NA returned.
truncate	If truncate=TRUE, any lengths beyond the end of the transcript or protein will be set to the last residue
protein.ids	A vector of protein.ids (or a RangedData object of proteins returned from another xmapcore function)

Details

If called with `as.vector=FALSE`, all locations which were not found, or are not applicable will be dropped from the result. This is because `RangedData` objects cannot represent NA or missing values.

Author(s)

Tim Yates

See Also

[xmapcore.to](#)
[xmapcore.details](#)
[xmapcore.all](#)
[xmapcore.range](#)
[xmapcore.filters](#)

Examples

```
if(interactive()) {
  # Get the gene for tp53
  gene = symbol.to.gene( tp53, as.vector=F )
  # And the transcripts for this gene
  transcripts = gene.to.transcript( symbol.to.gene( tp53 ) )
  # And the proteins for this transcript
  proteins = transcript.to.protein( transcripts )

  # get the transcript coords for the transcripts of this gene, at the start of this gene
  genome.to.transcript.coords( start( gene ), transcripts )
  #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
  genome.to.transcript.coords( start( gene ), transcripts, as.vector=FALSE )
  # RangedData with 1 row and 1 value column across 1 space
  #           space           ranges | coord.space
  #           <character>           <IRanges> | <character>
  # 1 ENST00000413465 [1018, 1018] | transcript

  genome.to.protein.coords( start( gene ), proteins )
  # 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
genome.to.protein.coords( start( gene ), proteins, as.vector=FALSE )
# 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
}
```

xmapcore.details	<i>xmapcore\`details\` functions</i>
------------------	--------------------------------------

Description

Get detailed annotations for the specified features.

Usage

```
array.details( ids, as.data.frame=FALSE )
chromosome.details( ids, as.data.frame=FALSE )
domain.details( ids, as.data.frame=FALSE )
est_exon.details( ids, as.data.frame=FALSE )
est_gene.details( ids, as.data.frame=FALSE )
est_transcript.details( ids, as.data.frame=FALSE )
exon.details( ids, as.data.frame=FALSE )
gene.details( ids, as.data.frame=FALSE )
prediction_transcript.details( ids, as.data.frame=FALSE )
probe.details( ids, as.data.frame=FALSE )
probeset.details( ids, as.data.frame=FALSE )
protein.details( ids, as.data.frame=FALSE )
synonym.details( ids, as.data.frame=FALSE )
transcript.details( ids, as.data.frame=FALSE )
```

Arguments

<code>ids</code>	Database identifiers for the features of interest
<code>as.data.frame</code>	If FALSE, data will be converted to a RangedData object if possible, otherwise a data.frame

Value

Results in an [RangedData](#) 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

[xmapcore.to](#)
[xmapcore.all](#)
[xmapcore.range](#)
[xmapcore.utils](#)
[xmapcore.filters](#)
[RangedData](#)

Examples

```

if(interactive()) {
  xmap.connect()
  gene.details(symbol.to.gene("TP53"))
}

```

xmapcore.env

xmapcore\'env\' functions

Description

Functions to access internal parameters

Usage

```

xmap.env()
xmap.get.param( key )
xmap.set.param( ... )
xmap.array.type( name=NULL, pick.default=FALSE )

```

Arguments

...	A list of key-value parameters you wish to set.
key	The key for the value you want to return.
name	The name of the array you want to use. If NULL, you will be presented with a menu of options.
pick.default	If TRUE, xmap.array.type will choose the first available arraytype for this species.

Details

These functions allow some access to xmapcore\'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 specified for the probe mappings. xmap.array.type allows a different type of array to be specified. This included for future compatibility.

Author(s)

Tim Yates Crispin J. Miller

See Also

[xmapcore.to](#)
[xmapcore.details](#)
[xmapcore.all](#)
[xmapcore.range](#)
[xmapcore.filters](#)

Examples

```
if(interactive()) {
  xmap.env()
  xmap.get.param( "debug" )
  xmap.connect()
  xmap.set.param( debug=TRUE)
  xmap.connect()
  xmap.set.param( debug=FALSE)
  xmap.disconnect()
}
## Not run:
  xmap.array.type("HuEx-1_0")

## End(Not run)
```

xmapcore.filters	<i>xmapcore\`filter\` functions</i>
------------------	-------------------------------------

Description

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

Usage

```
exonic( probesets, exclude=FALSE )
has.probes( probesets, num.probes=4, exclude=FALSE )
has.probes.atleast( probesets, num.probes=4, exclude=FALSE )
has.probes.in( probesets, num.probes=c( 1, 2, 3, 4 ), exclude=FALSE )
has.probes.between( probesets, min.probes=1, max.probes=4, exclude=FALSE, inclusive=TRUE )
intergenic( probesets, exclude=FALSE )
intronic( probesets, exclude=FALSE )
is.exonic( probesets )
is.intergenic( probesets )
is.intronic( probesets )
is.unreliable( 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 completely. 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

[xmapcore.to](#)
[xmapcore.details](#)
[xmapcore.all](#)
[xmapcore.range](#)
[xmapcore.filters](#)

Examples

```
if(interactive()){  
  xmap.connect()  
  ps <- gene.to.probeset(symbol.to.gene("TP53"))  
  exonic(ps)  
  intronic(ps)  
  intergenic(ps)  
  unreliable(ps)  
  is.exonic(ps)
```

```

    is.intronic(ps)
    is.intergenic(ps)
    is.unreliable(ps)
    has.probes(ps)
    has.probes.in(ps,1:3)
    has.probes.between(ps,2,3)
    has.probes.atleast(ps,4)
  }

```

xmapcore.range	<i>xmapcore \range\ functions</i>
----------------	-----------------------------------

Description

Get the features within the specified genome coordinates.

Usage

```

domains.in.range( chr, start, end, strand, as.vector=TRUE )
est_exons.in.range( chr, start, end, strand, as.vector=TRUE )
est_genes.in.range( chr, start, end, strand, as.vector=TRUE )
est_transcripts.in.range( chr, start, end, strand, as.vector=TRUE )
exons.in.range( chr, start, end, strand, as.vector=TRUE )
genes.in.range( chr, start, end, strand, as.vector=TRUE )
prediction_transcripts.in.range( chr, start, end, strand, as.vector=TRUE )
probesets.in.range( chr, start, end, strand, as.vector=TRUE )
probes.in.range( chr, start, end, strand, as.vector=TRUE )
proteins.in.range( chr, start, end, strand, as.vector=TRUE )
transcripts.in.range( chr, start, end, strand, as.vector=TRUE )
xmap.range.apply( x, f, filter=c( chr="space", start="start", end="end", strand="strand" ), coerce=c(

```

Arguments

as.vector	If TRUE returns a vector of database identifiers. If FALSE returns a RangedData object containing detailed annotation.
chr	The chromosome name (as a character)
start	Start of the region
end	End of the region
strand	1 == top stand, -1 == bottom strand
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 to \f\

Details

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

The function `xmap.range.apply` makes it possible to map any of these functions down the rows of a [RangedData](#) object. The defaults are set up so that it will handle the output of one of the `.in.range` 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 an [RangedData](#) 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

[xmapcore.to](#)
[xmapcore.details](#)
[xmapcore.all](#)
[xmapcore.utils](#)
[xmapcore.filters](#)
[RangedData](#)

Examples

```
if(interactive()) {
  xmap.connect()
  r <- genes.in.range( 17, 7510000, 7550000, 1 )
  genes.in.range( 17, 7510000, 7550000, -1 )
  xmap.range.apply(symbol.to.gene("TP53"),as.vector=FALSE),probes.in.range)
}
```

xmapcore.to

xmapcore 'to' functions

Description

Map between the different levels of annotation in X:Map. For example, given a vector of gene identifiers, `gene.to.exon` will return the exons in those genes.

Usage

```
array.to.probeset( ids, as.vector=TRUE )
domain.to.gene( ids, as.vector=TRUE )
domain.to.probeset( ids, as.vector=TRUE )
domain.to.protein( ids, as.vector=TRUE )
domain.to.transcript( ids, as.vector=TRUE )
est_exon.to.est_gene( ids, as.vector=TRUE )
est_exon.to.est_transcript( ids, as.vector=TRUE )
est_exon.to.probeset( ids, as.vector=TRUE )
est_gene.to.est_exon( ids, as.vector=TRUE )
est_gene.to.est_transcript( ids, as.vector=TRUE )
est_gene.to.probeset( ids, as.vector=TRUE )
est_transcript.to.est_exon( ids, as.vector=TRUE )
est_transcript.to.est_gene( ids, as.vector=TRUE )
est_transcript.to.probeset( ids, as.vector=TRUE )
exon.to.gene( ids, as.vector=TRUE )
exon.to.probeset( ids, as.vector=TRUE )
exon.to.transcript( ids, as.vector=TRUE )
gene.to.domain( ids, as.vector=TRUE )
gene.to.exon( ids, as.vector=TRUE )
gene.to.exon_probeset( ids, probes.min=4 )
gene.to.exon.probeset.expr( x, ids, probes.min=4 )
gene.to.probeset( ids, as.vector=TRUE )
gene.to.protein( ids, as.vector=TRUE )
gene.to.symbol( ids )
gene.to.synonym( ids, as.vector=TRUE )
gene.to.transcript( ids, as.vector=TRUE )
prediction_transcript.to.prediction_exon( ids )
prediction_transcript.to.probeset( ids, as.vector=TRUE )
probe.to.hit( ids )
probe.to.probeset( ids, as.vector=TRUE )
probeset.to.cdna_transcript( ids, as.vector=TRUE, rm.unreliable=TRUE )
probeset.to.domain( ids, as.vector=TRUE, rm.unreliable=TRUE )
probeset.to.est_exon( ids, as.vector=TRUE, rm.unreliable=TRUE )
probeset.to.est_gene( ids, as.vector=TRUE, rm.unreliable=TRUE )
probeset.to.est_transcript( ids, as.vector=TRUE, rm.unreliable=TRUE )
probeset.to.exon( ids, as.vector=TRUE, rm.unreliable=TRUE )
probeset.to.gene( ids, as.vector=TRUE, rm.unreliable=TRUE )
probeset.to.hit( ids, rm.unreliable=TRUE )
probeset.to.prediction_transcript( ids, as.vector=TRUE, rm.unreliable=TRUE )
probeset.to.probe( ids, as.vector=TRUE )
probeset.to.protein( ids, as.vector=TRUE, rm.unreliable=TRUE )
probeset.to.transcript( ids, as.vector=TRUE, rm.unreliable=TRUE )
protein.to.domain( ids, as.vector=TRUE )
protein.to.gene( ids, as.vector=TRUE )
protein.to.probeset( ids, as.vector=TRUE )
protein.to.transcript( ids, as.vector=TRUE )
symbol.to.est_gene( ids, as.vector=TRUE )
```

```

symbol.to.est_transcript( ids, as.vector=TRUE )
symbol.to.gene( ids, as.vector=TRUE )
symbol.to.transcript( ids, as.vector=TRUE )
synonym.to.est_gene( ids, as.vector=TRUE )
synonym.to.est_transcript( ids, as.vector=TRUE )
synonym.to.gene( ids, as.vector=TRUE )
synonym.to.transcript( ids, as.vector=TRUE )
transcript.to.cdnaprobeset( ids, as.vector=TRUE )
transcript.to.domain( ids, as.vector=TRUE )
transcript.to.exon( ids, as.vector=TRUE )
transcript.to.exon_probeset( ids, probes.min=4 )
transcript.to.gene( ids, as.vector=TRUE )
transcript.to.probeset( ids, as.vector=TRUE )
transcript.to.protein( ids, as.vector=TRUE )
transcript.to.synonym( ids, as.vector=TRUE )
transcript.to.translatedprobes( ids )

```

Arguments

as.vector	If TRUE returns a vector of database identifiers. If FALSE returns a link{RangedData} object containing detailed annotation.
ids	Database identifiers to map from. Can be either a vector of database identifiers, or a RangedData 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-explanatory. 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 `transcript.to.translatedprobes` always returns a `RangedData` object containing each probe that hits the specified translated transcripts and the start and end locations of that hit relative to the start of those transcripts.

Value

Results in an [RangedData](#) object, one 'row' per feature, containing detailed annotations, or a vector, as defined by `as.vector`.

Author(s)

Tim Yates

See Also

[xmapcore.details](#)
[xmapcore.all](#)
[xmapcore.range](#)
[xmapcore.utils](#)
[xmapcore.filters](#)
[link{RangedData}](#)

Examples

```

if(interactive()) {
  xmap.connect()
  gene.to.exon(symbol.to.gene("TP53"))
}

```

xmapcore.utils	<i>xmapcore\utils\functions</i>
----------------	---------------------------------

Description

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

Usage

```

xmap.connect( name )
xmap.disconnect()
xmap.toggle.caching()
xmap.clear.cache()
xmap.gene.plot( genes, style=c( real, stacked, jointed, injointed ), rect.func=.default.plotter )

```

Arguments

name	The name of the database to connect to.
genes	A vector of Gene Names, or a RangedData object returned from another xmapcore call
style	The style of graph you require
rect.func	For each transcript in the gene, this will be called and passed a data.frame containing gene, transcript, exon, and rectangle data for you to do your own drawing

Details

xmap.connect is used to establish a connection to an instance of the X:Map database, and xmap.disconnect closes the connection.

Many of the functions in xmapcore cache results locally. The function xmap.toggle.caching turns this functionality on and off, and xmap.clear.cache 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.

Author(s)

Tim Yates Crispin J. Miller

See Also

[xmapcore.to](#)
[xmapcore.details](#)
[xmapcore.all](#)
[xmapcore.range](#)
[xmapcore.filters](#)

Examples

```
if(interactive()) {
  xmap.connect()
  xmap.toggle.caching()
  xmap.toggle.caching()

  #NOTE: since the next function empties out the local cache, dont
  #run it unless you want to do this!
  #xmap.clear.cache()
}
```

xmapcore.utr

xmapcore coding functions

Description

Functions to deal with coding regions and UTRs

Usage

```
transcript.to.utr.range( ids, end=c( "both", "5", "3" ), as.data.frame=FALSE )
transcript.to.coding.range( ids, end=c( "both", "5", "3" ), as.data.frame=FALSE )
utr.probesets( probesets, transcripts, end=c( "both", "5", "3" ) )
coding.probesets( probesets, transcripts, end=c( "both", "5", "3" ) )
```

Arguments

ids	A vector of Transcript Names, or a RangedData object of Transcripts returned from another xmapcore call.
as.data.frame	If FALSE, data will be converted to a RangedData object if possible, otherwise a data.frame
probesets	An optional vector of Probeset Names, or a RangedData object of Probesets returned from another xmapcore call.

transcripts	An optional vector of Transcript Names, or a RangedData object of Transcripts returned from another xmapcore call.
end	Which end ("both", "3" or "5") of the Transcript(s) you are interested in (defaults to both).

Details

The first two functions given here, `transcript.to.utr.range` and `transcript.to.coding.range` return the transcripts of interest, with their ranges adjusted depending on the UTR of each.

With `transcript.to.utr.range`, 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 `transcript.to.coding.range` function returns the same as calling `transcript.details`, 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.

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

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

A call to `utr.probesets` 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 `utr.probesets` 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 `coding.probesets` method does the inverse of the `utr.probesets` function: it returns probesets having at least one probe in the coding region of a Transcript (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 `utr.probesets` and `coding.probesets` 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.

Author(s)

Tim Yates

See Also

[xmapcore.to](#)
[xmapcore.details](#)
[xmapcore.all](#)
[xmapcore.range](#)

*xmapcore.filters***Examples**

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

Index

*Topic **package**

xmapcore-package, 2

all.arrays (xmapcore.all), 3
all.chromosomes (xmapcore.all), 3
all.domains (xmapcore.all), 3
all.est_exons (xmapcore.all), 3
all.est_genes (xmapcore.all), 3
all.est_transcripts (xmapcore.all), 3
all.exons (xmapcore.all), 3
all.genes (xmapcore.all), 3
all.prediction_transcripts
 (xmapcore.all), 3
all.probes (xmapcore.all), 3
all.probesets (xmapcore.all), 3
all.proteins (xmapcore.all), 3
all.symbols (xmapcore.all), 3
all.synonyms (xmapcore.all), 3
all.transcripts (xmapcore.all), 3
array.details (xmapcore.details), 6
array.to.probeset (xmapcore.to), 11

chromosome.details (xmapcore.details), 6
coding.probesets (xmapcore.utr), 15

domain.details (xmapcore.details), 6
domain.to.gene (xmapcore.to), 11
domain.to.probeset (xmapcore.to), 11
domain.to.protein (xmapcore.to), 11
domain.to.transcript (xmapcore.to), 11
domains.in.range (xmapcore.range), 10

est_exon.details (xmapcore.details), 6
est_exon.to.est_gene (xmapcore.to), 11
est_exon.to.est_transcript
 (xmapcore.to), 11
est_exon.to.probeset (xmapcore.to), 11
est_exons.in.range (xmapcore.range), 10
est_gene.details (xmapcore.details), 6
est_gene.to.est_exon (xmapcore.to), 11
est_gene.to.est_transcript
 (xmapcore.to), 11
est_gene.to.probeset (xmapcore.to), 11
est_genes.in.range (xmapcore.range), 10
est_transcript.details
 (xmapcore.details), 6
est_transcript.to.est_exon
 (xmapcore.to), 11
est_transcript.to.est_gene
 (xmapcore.to), 11
est_transcript.to.probeset
 (xmapcore.to), 11
est_transcripts.in.range
 (xmapcore.range), 10
exon.details (xmapcore.details), 6
exon.to.gene (xmapcore.to), 11
exon.to.probeset (xmapcore.to), 11
exon.to.transcript (xmapcore.to), 11
exonic (xmapcore.filters), 8
exons.in.range (xmapcore.range), 10
ExpressionSet, 13

gene.details (xmapcore.details), 6
gene.to.domain (xmapcore.to), 11
gene.to.exon (xmapcore.to), 11
gene.to.exon_probeset (xmapcore.to), 11
gene.to.probeset (xmapcore.to), 11
gene.to.protein (xmapcore.to), 11
gene.to.symbol (xmapcore.to), 11
gene.to.synonym (xmapcore.to), 11
gene.to.transcript (xmapcore.to), 11
genes.in.range (xmapcore.range), 10
genome.to.protein.coords
 (xmapcore.coords), 4
genome.to.transcript.coords
 (xmapcore.coords), 4

has.probes (xmapcore.filters), 8
intergenic (xmapcore.filters), 8

- intronic (xmapcore.filters), 8
- IRanges, 3
- is.exonic (xmapcore.filters), 8
- is.intergenic (xmapcore.filters), 8
- is.intronic (xmapcore.filters), 8
- is.unreliable (xmapcore.filters), 8
- prediction_transcript.details
 - (xmapcore.details), 6
- prediction_transcript.to.prediction_exon
 - (xmapcore.to), 11
- prediction_transcript.to.probeset
 - (xmapcore.to), 11
- prediction_transcripts.in.range
 - (xmapcore.range), 10
- probe.details (xmapcore.details), 6
- probe.to.hit (xmapcore.to), 11
- probe.to.probeset (xmapcore.to), 11
- probes.in.range (xmapcore.range), 10
- probeset.details (xmapcore.details), 6
- probeset.to.cdna_transcript
 - (xmapcore.to), 11
- probeset.to.domain (xmapcore.to), 11
- probeset.to.est_exon (xmapcore.to), 11
- probeset.to.est_gene (xmapcore.to), 11
- probeset.to.est_transcript
 - (xmapcore.to), 11
- probeset.to.exon (xmapcore.to), 11
- probeset.to.gene (xmapcore.to), 11
- probeset.to.hit (xmapcore.to), 11
- probeset.to.prediction_transcript
 - (xmapcore.to), 11
- probeset.to.probe (xmapcore.to), 11
- probeset.to.protein (xmapcore.to), 11
- probeset.to.transcript (xmapcore.to), 11
- probesets.in.range (xmapcore.range), 10
- protein.coords.to.genome
 - (xmapcore.coords), 4
- protein.details (xmapcore.details), 6
- protein.to.domain (xmapcore.to), 11
- protein.to.gene (xmapcore.to), 11
- protein.to.probeset (xmapcore.to), 11
- protein.to.transcript (xmapcore.to), 11
- proteins.in.range (xmapcore.range), 10
- RangedData, 3, 4, 6, 7, 10, 11, 13
- symbol.to.est_gene (xmapcore.to), 11
- symbol.to.est_transcript (xmapcore.to), 11
- symbol.to.gene (xmapcore.to), 11
- symbol.to.transcript (xmapcore.to), 11
- synonym.details (xmapcore.details), 6
- synonym.to.est_gene (xmapcore.to), 11
- synonym.to.est_transcript
 - (xmapcore.to), 11
- synonym.to.gene (xmapcore.to), 11
- synonym.to.transcript (xmapcore.to), 11
- transcript.coords.to.genome
 - (xmapcore.coords), 4
- transcript.details (xmapcore.details), 6
- transcript.to.cdna_probeset
 - (xmapcore.to), 11
- transcript.to.coding.range
 - (xmapcore.utr), 15
- transcript.to.domain (xmapcore.to), 11
- transcript.to.exon (xmapcore.to), 11
- transcript.to.exon_probeset
 - (xmapcore.to), 11
- transcript.to.gene (xmapcore.to), 11
- transcript.to.probeset (xmapcore.to), 11
- transcript.to.protein (xmapcore.to), 11
- transcript.to.synonym (xmapcore.to), 11
- transcript.to.translated_probes
 - (xmapcore.to), 11
- transcript.to.utr.range (xmapcore.utr), 15
- transcripts.in.range (xmapcore.range), 10
- unreliable, 13
- unreliable (xmapcore.filters), 8
- utr.probesets (xmapcore.utr), 15
- xmap.array.type (xmapcore.env), 7
- xmap.clear.cache (xmapcore.utils), 14
- xmap.connect (xmapcore.utils), 14
- xmap.disconnect (xmapcore.utils), 14
- xmap.env (xmapcore.env), 7
- xmap.gene.plot (xmapcore.utils), 14
- xmap.get.param (xmapcore.env), 7
- xmap.range.apply (xmapcore.range), 10
- xmap.set.param (xmapcore.env), 7
- xmap.toggle.caching (xmapcore.utils), 14
- xmapcore (xmapcore-package), 2
- xmapcore-package, 2

xmapcore.all, [3](#), 5, 7–9, 11, 14–16
xmapcore.coords, [4](#)
xmapcore.details, 4, 5, [6](#), 8, 9, 11, 14–16
xmapcore.env, [7](#)
xmapcore.filters, 4, 5, 7, [8](#), 8, 9, 11, 14, 15,
17
xmapcore.range, 4, 5, 7–9, [10](#), 14–16
xmapcore.to, 4, 5, 7–9, [11](#), 11, 15, 16
xmapcore.utils, 4, 7, 11, [14](#), 14
xmapcore.utr, [15](#)