xmapcore

April 20, 2011

---

xmapcore.all   *xmapcore állfunctions*

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

1

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.details   *xmapcore detailsfunctions*

Description

Get detailed annotations for the specified features.

Usage

```
array.details( ids )
chromosome.details( ids )
domain.details( ids )
est_exon.details( ids )
est_gene.details( ids )
est_transcript.details( ids )
exon.details( ids )
gene.details( ids )
prediction_transcript.details( ids )
probe.details( ids )
probeset.details( ids )
protein.details( ids )
synonym.details( ids )
transcript.details( ids )
```

Arguments

ids   Database identifiers for the features of interest

Value

Results in an RangedData object, 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 énvfunctions*

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 specfied for the probe mappings. xmap.array.type allows a different type of array to be specfied. 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   *xmapcore ́filter functions*

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, incl
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 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

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-package   *Provide access to the X:Map annotation database*

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:

xmaprotein: 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

xmaprotein xmapbridge IRanges

---

xmapcore.range   *xmapcore ŕangéfunctions*

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", stran
```

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.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.cdnatranscript( 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 )
```

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 indentifiers,

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.

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).

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   *xmapcore útilsfunctions*

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' ), r
```

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, don't
  #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" ) )
transcript.to.coding.range( ids, end=c( "both", "5", "3" ) )
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.
`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 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 `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