GeneR
November 11, 2009

R topics documented:

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>GeneR</td>
<td>2</td>
</tr>
<tr>
<td>globalSeg</td>
<td>3</td>
</tr>
<tr>
<td>Globals Variables</td>
<td>5</td>
</tr>
<tr>
<td>getOrfs</td>
<td>6</td>
</tr>
<tr>
<td>and.globalSeg</td>
<td>8</td>
</tr>
<tr>
<td>and.segSet</td>
<td>9</td>
</tr>
<tr>
<td>appendSeq</td>
<td>10</td>
</tr>
<tr>
<td>as.globalSeg</td>
<td>11</td>
</tr>
<tr>
<td>as.segSet</td>
<td>12</td>
</tr>
<tr>
<td>assemble</td>
<td>13</td>
</tr>
<tr>
<td>AtoR</td>
<td>14</td>
</tr>
<tr>
<td>bankDensityProfile</td>
<td>15</td>
</tr>
<tr>
<td>bankSummary</td>
<td>16</td>
</tr>
<tr>
<td>CompoSeq</td>
<td>17</td>
</tr>
<tr>
<td>Concat</td>
<td>18</td>
</tr>
<tr>
<td>deleteCR</td>
<td>19</td>
</tr>
<tr>
<td>densityProfile</td>
<td>20</td>
</tr>
<tr>
<td>DnaToRna</td>
<td>22</td>
</tr>
<tr>
<td>exactWord</td>
<td>23</td>
</tr>
<tr>
<td>fastaDescription</td>
<td>24</td>
</tr>
<tr>
<td>Buffers</td>
<td>25</td>
</tr>
<tr>
<td>getAccn</td>
<td>26</td>
</tr>
<tr>
<td>getParam</td>
<td>27</td>
</tr>
<tr>
<td>getSeq</td>
<td>27</td>
</tr>
<tr>
<td>indexFasta</td>
<td>28</td>
</tr>
<tr>
<td>insertSeq</td>
<td>29</td>
</tr>
<tr>
<td>lowerSeq</td>
<td>30</td>
</tr>
<tr>
<td>makeIndex</td>
<td>30</td>
</tr>
<tr>
<td>mask</td>
<td>32</td>
</tr>
<tr>
<td>Match</td>
<td>33</td>
</tr>
<tr>
<td>not.globalSeg</td>
<td>34</td>
</tr>
<tr>
<td>not.segSet</td>
<td>35</td>
</tr>
<tr>
<td>or.globalSeg</td>
<td>36</td>
</tr>
<tr>
<td>or.segSet</td>
<td>37</td>
</tr>
<tr>
<td>placeString</td>
<td>38</td>
</tr>
<tr>
<td>plot.globalSeg</td>
<td>39</td>
</tr>
<tr>
<td>posMaskedSeq</td>
<td>40</td>
</tr>
</tbody>
</table>
**Description**

GeneR packages allow direct use of nucleotide sequences within R software. Functions can be used to read and write sequences from main file formats (Embl, Genbank and Fasta) in order to perform a lot of manipulations and analyses. Main functions are implemented with C extensions.

**Details**

To summarize, we can split major functions into 4 categories.

1. **Reading and writing sequences**

GeneR has been designed for fast sequence retrieving even from very large sequence databanks, in Fasta, Embl or Genbank formats. It is also possible to enter sequences directly from a R command line.

There are two ways to store sequences:

- In a C adapted class (buffers) that stores in addition some globals variables, like working strand, size of original sequence and so on.
  
  It is useful when, for example, we have to work on a subset of a whole chromosome (i.e. a gene). In this case it will be worthwhile to load only the gene in R. Nevertheless, it will remain easy to associate positions on chromosome and positions on gene . . .

  A complete description on this C class is given in page `globals`.

- As a character string, the more logical way to store short sequences like "ATGTCGTG". It concerns all functions like "strxxx" (strComp, strMultiExtract etc.).
II- Handling sequences
The usual copy-paste of parts of sequences or other manipulations can be performed by functions using vectors of strands and positions. Annotations from the features field within formatted sequence entries can be extracted and used directly in vectors. By this way, it is easy to extract sequence fragments of interest.

III- Analyzing sequences
Some tools are designed to count oligo-nucleotides, to look for exact word positions or to shuffle sequences (useful for statistical validations).

IV- Genetic tools
Finally, the package also contains functions related to genetic and structural aspects of the sequences: ORF localization, translation, or RNA secondary structure determination (with extension of GeneR: GeneRfold package).

See Also
globals, readSeq, readEmblLocation, getOrfs

Examples

```r
## First of all you can try
demo(GeneR)
```
globalSeg

- globalSeg: a list of segSet. It allows the notion of list of discontinuous segments (our use: a list of gene’s models as a gene’s model is stored as a list of its exons). In our sample, globalSeg will be the list of the 3 regions A,B and C. Note that it store more information than just a matrix with 2 columns containing all segments of theses 3 regions.

For a better comprehension of other man pages, we introduce this notation:

- a segment is just a part of a line determined by two values (from and to)
- an object of class segSet is a set of \( n \) segments, determined by a matrix \( n \times 2 \)
- an object of class globalSeg is a set of segSets, determined by a list of matrix.

Author(s)

Epissage group at CGM.

See Also

See also as.segSet, as.globalSeg, unionSeg

Examples

```r
a = list(
  matrix( c( 1, 15, 17, 5, 45, 38),ncol=2),
  matrix( c(100, 120),ncol=2),
  matrix( c(130, 135, 140, 145),ncol=2),
  matrix( c(142, 160),ncol=2))

b = list(
  matrix( c(15, 28, 18, 45),ncol=2),
  matrix( c(1, 15, 25, 10, 20, 40),ncol=2),
  matrix( c(17, 35, 23, 38),ncol=2),
  matrix( c(100, 110, 105, 120),ncol=2))

a = as.globalSeg(a)
b = as.globalSeg(b)

par(mar=c(1,0,1,0))
par(mfrow=c(8,1))
plot(a,xlim=c(1,160),main="A")
plot(b,xlim=c(1,160),main="B")
plot(and(b),xlim=c(1,160),main="and(B)")
plot(or(b),xlim=c(1,160),main="or(B)")
plot(Xor(b),xlim=c(1,160),main="Xor(B)")
plot(a&b,xlim=c(1,160),main="A&B")
plot(a|b,xlim=c(1,160),main="A|B")
plot(Xor(a,b),xlim=c(1,160),main="Xor(A,B)")
```
Globals Variables

Description

There are two ways to store sequences in GeneR:

• In a C adapted class (buffers) that stores in addition some globals variables, like working strand, size of original sequence and so on.
  It is useful when, for example, we have to work on a subset of a whole chromosome (i.e. a gene). In this case it will be worthwhile to load only the gene in R. Nevertheless, it will remain easy to associate positions on chromosome and positions on gene . . .

• As a character string, the more logical way to store short sequences like "ATGTCGTG". It concerns all functions like "strxxx" (strComp, strReadFasta etc.).

Details

When GeneR load a subset of a larger sequence stored in a bank file, it will store the following informations in the C adapted class (buffers, by default 100 buffers than can be extended if necessary):

• subsequence (i.e. the succession of A,T,G,C).
• positions of the extremities of the subsequence in the master sequence
• size of the whole sequence in the bank file
• name of the sequence

For specific purposes as renaming a sequence, all these variables can be viewed and carefully changed at any time (here functions getAccn and setAccn).

Several sequences can be stored simultaneously and called by their buffer number.

Strand is another global variable which can be set and viewed (functions getStrand and setStrand).

It is used as input parameter in many functions to analyze complementary strand. It was designed to avoid doing explicitly the complement of the loaded strand then to store it in a buffer with, as consequence, loss of the informations linked to the master sequence.

We have defined 3 types of addresses on a subsequence extracted from a master sequence:

• Absolute addresses i.e. addresses on the master sequence, from the 5’ end of the input strand referred as forward (noted A)
• Real addresses, i.e. addresses on the master sequence, from the 5’ end of one of strands (noted R)
• Relative addresses, i.e. addresses on working subsequence, from the 5’ end of one of strands (noted T).

Let’s show an example, if we read sequence from 11 to 20 from a gene of size 40:

<table>
<thead>
<tr>
<th>Strand 0 (Forward strand)</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 20 40</td>
</tr>
<tr>
<td>1 11 20 40</td>
</tr>
<tr>
<td>1 10</td>
</tr>
</tbody>
</table>

xxxxxxATGTCGTGTAxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
getOrfs

<table>
<thead>
<tr>
<th>10</th>
<th>1</th>
<th>Relative (T)</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>30</td>
<td>21</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>20</td>
</tr>
</tbody>
</table>

1 Real (R)

40 Absolute (A)

Strand 1 (Reverse strand)

Obviously, when an entire sequence is stored, real and relative addresses will be the same.

Although all functions using positions need and return absolute addresses, 6 functions allow to convert R, A, T into any other type (functions RtoA, RtoA, RtoA, RtoA, RtoA, RtoA).

A global variable strand is used to convert positions (see setStrand getStrand).

See Also

AtoT, AtoR, RtoA, RtoT, TtoR, setStrand, getStrand, setParam, getAccn, setAccn

Examples

```
## Make a dummy sequence
s <- "xxxxxxxxxxxxATGTGTCGTAxxxxxxxxxxxxxxxxxxxx"
placeString(s)
writeFasta(file="toto.fa")

indexFasta("toto.fa")
readFasta("toto.fa",from=11,to=20)

getParam()
## $begin
## [1] 11
## $size
## [1] 40
## $strand
## [1] 0
## [...]

## With strand = 0
TtoA(c(1,10))
##[1] 10 19

TtoR(c(1,10))
##[1] 10 19
```

---

**getOrfs**

*Gets ORFs from a sequence*

**Description**

Gets ORFs (Open Reading Frames) from a sequence.
getOrfs

Usage

getOrfs(phase = NULL, seqno = 0, start = "atg",
         stop = c("taa", "tag", "tga"), complete = TRUE, suborfs = TRUE)
maxOrf(seqno = 0, phase = NULL, start = "atg",
       stop = c("taa", "tag", "tga"), complete = TRUE)

Arguments

seqno  Integer/scalar, Sequence number (buffer number)
phase  Integer/scalar, 1, 2 or 3, NULL for all three phases
start  string/vector, start codons
stop   string/vector, stop codons
complete  Flag: true returns only complet Orfs, else return all Orfs.
suborfs  Flag: true returns all orfs including subparts of a large orf if it exists "atg" in the
          phase. False: does not returns sub-orfs.

Value

getOrf returns a table of positions. NULL if no Orfs.
maxOrf returns the size of the largest Orf, -1 if no Orf.
All functions return NA if error.

Note

Reverse strand: not implemented

Author(s)

A. Lucas, Emna Marrakchi and Vincent Lefort

Examples

s <- "gtcatgcatgctaggtgacagttaaaatgcgtctaggtgacagtctaacaa"
placeString(s)
getOrfs(phase = NULL, seqno = 0)
maxOrf()

# To get all ORFs on the reverse strand:
sc <- getSeq(0, 1)
placeString(sc, seqno = 1)
getOrfs(phase = NULL, seqno = 1)

# All orfs on both strands:
rbind(getOrfs(seqno = 0), getOrfs(seqno = 1))
**Description**

Computes intersection of two objects of class `globalSeg` a and b, i.e. returns segments that are both in a and b.

When used with only one object of class `globalSeg`, it computes intersection of all its segments set (an segments sets is a set of segments; an object of class `globalSeg` is a set of segments sets). In this case, when `global=TRUE`, it computes the intersection of all the segments (see example below).

**Usage**

```r
and.globalSeg(a, b = NULL, global = FALSE, byrows = FALSE, relist = TRUE, ...)
```

**Arguments**

- `a, b`: elements of class `globalSeg`
- `global`: used if `b=NULL`
- `byrows`: if `TRUE`, a and b must be lists of same length, and intersection is computed for each elements from a with its homologue in b.
- `relist`: is `TRUE`, results are re-listed (envelops are union of all a and b).
- `...`: Unused

**Value**

an element of class `globalSeg`.

**Author(s)**

Odile Rogier

**See Also**

- `globalSeg`
- `and.segSet`

**Examples**

```r
a = list(
  matrix( c( 1, 15, 17, 5, 45, 38),ncol=2),
  matrix( c( 100 , 120),ncol=2),
  matrix( c( 130, 135, 140, 145),ncol=2),
  matrix( c( 142 , 160),ncol=2))

b = list(
  matrix( c(15, 28, 18, 45),ncol=2),
  matrix( c(1, 15, 25, 10, 20, 40),ncol=2),
  matrix( c(17, 35, 23, 38),ncol=2),
  matrix( c(100, 110, 105, 120),ncol=2))
```
```r
a = as.globalSeg(a)
b = as.globalSeg(b)

c = and(a, b, byrows=TRUE)
par(mfrow=c(5,1))
plot(a, xlim=c(1,160), main="A")
plot(b, xlim=c(1,160), main="B")
plot(a&b, xlim=c(1,160), main="A&B")
plot(c, xlim=c(1,160), main="A&B, byrow=T")
plot(and(a), xlim=c(1,160), main="and(A)")

## Show result
a&b
c
and(a)
```

### and.segSet

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>and(a,b) Computes intersection of two objects of class segSet a and b, i.e. returns segments that are both in a and b.</td>
</tr>
<tr>
<td>When used with only one object of class segSet it returns the common segment of all segments of element 'a'.</td>
</tr>
</tbody>
</table>

### Usage

```r
and.segSet(a, b = NULL, ...)
```

### Arguments

- `a, b` elements of class segSet
- `...` Unused

### Value

- an element of class segSet.

### Author(s)

Antoine Lucas

### See Also

- `globalSeg`, `and.globalSeg`
Examples

```
a = matrix(c(1,5,15,45,17,38,
          100,120,130,140,
          135,145,142,160),
         ncol=2,byrow=TRUE)
b = matrix(c(15,18, 28,45,
          1,10, 15,20, 25,40,
          17,23, 35,38,100,105,
          110,120),ncol=2,byrow=TRUE)
a <- as.segSet(a)
b <- as.segSet(b)
c = and(a,b)
par(mfrow=c(3,1))
plot(a,xlim=c(1,160))
plot(b,xlim=c(1,160))
plot(c,xlim=c(1,160))
## Another sample
a = matrix(c(1,30,40,50,60,70,80,110),ncol=2,byrow=TRUE)
b = matrix(c(1,10,20,30,40,70,80,90,100,110),ncol=2,byrow=TRUE)
a <- as.segSet(a)
b <- as.segSet(b)
c = and(a,b)
par(mfrow=c(3,1))
plot(a,xlim=c(1,160),main="A")
plot(b,xlim=c(1,160),main="B")
plot(c,xlim=c(1,160),main="A&B")
## Show result
c
and(a)
```

appendSeq

**Appends two sequences**

Description

appends a sequence at the end of another one

Usage

```
appendSeq(destSeqno=0, seqno=1)
```

Arguments

- **destSeqno**: Sequence number of sequence to be appended
- **seqno**: Sequence number of sequence to add at the end of destSeqno
as.globalSeg

Value

0: no error; NULL if error and a warning if problem in memory allocation.

Note

This function frees the reverse complementary of sequence destSeqno

Author(s)

Antoine Lucas

See Also

globalSeq, concat, assemble

Examples

s <- "CTCGTGGTGAAG"
placeString(s, seqno=0)
placeString(s, seqno=1)
appendSeq(0,1)
getSeq(0)

as.globalSeg  GlobalSeq manipulation

Description

These functions implements standards routines for globalSeg manipulations.

Usage

as.globalSeg(segments)
as.matrix.globalSeg(x,...)
range.globalSeg(globalSeg, na.rm = FALSE, global=FALSE,...)

Arguments

x, globalSeg  Element of class globalSeg
segments  Element of class segments
na.rm  logical, indicating if `NA`'s should be omitted.
global  Flag, TRUE: range return one value for the whole list; FALSE: range return one
       value by element of the list.
...  Additional parameters

Value

an element of class globalSeg.
as.segSet

Description

These functions implement standard routines for segSet manipulations.

Usage

as.segSet(x)
as.matrix.segSet(x,...)
as.data.frame.segSet(x,row.names = NULL,optional = FALSE,...)
plot.segSet(x,...)

Arguments

x

Element of class segSet

row.names

'NULL' or a character vector giving the row names for the data frame. Missing values are not allowed.

optional

logical. If 'TRUE', setting row names and converting column names (to syntactic names) is optional.

...

Additional parameters

Value

an element of class segSet.

Author(s)

Antoine Lucas

See Also

globalSeg
assemble

**Concatenates parts of a sequence**

Description

concatenates parts of a sequence (in any strand) and puts resulting sequence in destination sequence buffer.

Usage

```r
assemble(seqno=0, from=1, to=0, strand=getStrand(), destSeqno=1)
```

Arguments

- `seqno` Integer/scalar, Sequence number (buffer number)
- `from, to` Integer/vector, Absolute addresses of the begin and the end of the fragments, (1 means the first nucleotide and 0 conventionally the last one; from must not be larger than to)
- `strand` Integer/vector, Strand (forward: 0, reverse: 1)
- `destSeqno` Output sequence number

Value

destSeqno or -1 if error.

Author(s)

L. Cottret

See Also

- `getSeq`
- `concat`
- `appendSeq`

Examples

```r
s <- "aaaaagctagcaagtactacctaccccctagctagtagatattt"
placeString(s)
x <- c(1, 21, 39)
y <- c(4, 28, 0)
assemble(from=x, to=y)
assemble(from=x, to=y, strand=c(0, 1, 0), destSeqno=2)
getSeq(1)
getSeq(2)
```
### AtoR

**Conversion of addresses**

**Description**

Converts addresses on a sequence according to the working strand and the address type (A: absolute, T: true, R: relative).

**Usage**

```
AtoR(x, segno=0)
AtoT(x, segno=0)
RtoA(x, segno=0)
RtoT(x, segno=0)
TtoA(x, segno=0)
TtoR(x, segno=0)
```

**Arguments**

- **x** Integer/vector. Addresses
- **segno** Integer/scalar, Sequence number (buffer number)

**Details**

All details on addresses and global variables are on the page `globals`.

**Value**

Integer vector with new addresses

**Note**

All results depend on the value of Strand. See: `setStrand`, `getStrand`.

**Author(s)**

L. Cottret

**See Also**

- `AtoT`, `RtoA`, `RtoT`, `TtoA`, `TtoR`, `setStrand`, `getStrand`

**Examples**

```r
s<="cgtagtagctagctagctagctagctagc"
placeString (s, segno=0)
# s of size 30
address <- c(4,20)

# On reverse strand:
setStrand(1)
```
bankDensityProfile

Computes density profile(s) of a bank of fasta sequences

Description

Computes profile(s) of user defined quantities from the beginning or the end of sequence fragments. Profile(s) is(are) constituted of bins of equal size with the mean, the standard deviation and the number of valid events for each bin.

Usage

bankDensityProfile (file, seqno=0, fun=seqSkew, fileout= NULL, nbin, sizeBin, allSeq=FALSE, fromEnd=FALSE, name = "all", threshold=0, strand=getStrand(),accu=FALSE,case="all")

Arguments

file Integer/scalar, File name of the bank (fasta file)
seqno Integer/scalar, Sequence number (buffer number)
strand Integer/scalar, Strand (forward: 0, reverse: 1)
fun Function, Function to be used (for example seqSkew
fileout String/scalar, If not NULL, a file to write results
nbin Integer/scalar, Number of bins to be created before the origin
sizeBin Integer/scalar, Size of the bins
allSeq Logical/scalar, If TRUE, Input sequence is the whole sequence, if FALSE, input is only the half sequence
fromEnd Logical/scalar, TRUE: Origin is the end of each sequence, if FALSE: Origin is the begining of each sequence
name String/vector, Names of sequences in bank file, "all": uses all sequences of the bank
threshold Integer/scalar, For each bin, maximum number of N tolerated in the sequence to participate to the computation
case String/scalar, Case of the letters taken into account ("all", "upper", "lower")
accu Flag, if true, returns sum , sum of square, and count on demanding function; else returns, mean and standard error on mean.
Value

A list of matrices, with the mean(s), the standard deviation(s) and the number of valid sub-fragments in each bin.

Author(s)

Emna Marrakchi and Antoine Lucas

See Also

densityProfile, bankSummary, GCcontent, seqSkew

Examples

```r
## We create 2 banks
for(i in 1:10) {
  s=randomSeq(n=100)
  placeString(s, seqno=0)
  writeFasta("toto_norm.fa", append=TRUE, name=i)

  s=randomSeq(prob=c(0.3,1,1,1,0)/3.3, n=100)
  placeString(s, seqno=0)
  writeFasta("toto_lowT.fa", append=TRUE, name=i)
}

densNorm <- bankDensityProfile("toto_norm.fa", nbin=10, sizeBin=10, allSeq=TRUE)
densLowT <- bankDensityProfile("toto_lowT.fa", nbin=10, sizeBin=10, allSeq=TRUE)

par(mfrow=c(1,2))
## Plot skew in normal bank
plot(densNorm$skta, main="TA skew Normal bank", ylim=c(-0.8, 0.3))
## Plot skew in low T bank
plot(densLowT$skta, main="TA skew low T bank", ylim=c(-0.8, 0.3))
## Show numbers
densLowT
```

---

**bankSummary**

*Computes informations on all a bank file*

**Description**

This functions computes informations (with function compoSeq, GCcontent, of seqSkew) on a complete bank file.

It returns a data frame with one line by sequences in the bank file.

Parameter "name" can be use to limit the exploration to only a few sequences.

**Usage**

```r
bankSummary(file, name = NULL, type = "F", fun = compoSeq, seqno = 0)
```
**CompoSeq**

**Arguments**

- **file**: Integer/scalar, File name of the bank (fasta file)
- **seqno**: Integer/Scalar, Sequence number (buffer number)
- **name**: String/vector, Names of sequences in bank file, NULL: uses all sequences of the bank
- **type**: String/scalar, Bank format ("F" -> fasta, "E" -> embl, "G" -> GenBank)
- **fun**: Function, Function to be used (for example seqSkew)

**Value**

a data frame with one line by sequence.

**Author(s)**

Antoine Lucas

**See Also**

bankDensityProfile

**Examples**

```r
for(i in 1:8) {
  s=randomSeq(n=100)
  placeString(s,seqno=0)
  writeFasta("toto_norm.fa",append=TRUE,name=i)
}
bankSummary(file="toto_norm.fa")
bankSummary(file="toto_norm.fa",fun=seqSkew)
```

---

**CompoSeq**

*Composition in mono, di or trinucleotides of a sequence*

**Description**

Gives composition in mono, di or trinucleotides of fragments of a sequence

Only A, T, G, C are counted, letter as U, X, S, M, N, are counted as 'X'. Flag "case" specify weather we take into account letters in lower, upper ar all case.

**Usage**

```r
compoSeq(seqno=0, from=1, to=0, strand=getStrand(),
  wsize=1, p=TRUE, case="all", step=wsize)
strCompoSeq(s, wsize=1, p=TRUE, case="all", step=wsize)
```
Concat

Arguments

- `seqno` Integer/scalar, Sequence number (buffer number)
- `s` String/scalar, A sequence as character string
- `wsize` Integer/scalar, size of k-uples.
- `from,to` Integer/scalar, Absolute addresses of the beginning and the end of the fragment, (1 means the first nucleotide and 0 conventionally the last one; from must not be larger than to and both vectors must be the same size)
- `strand` Integer/vector, Strand (forward: 0, reverse: 1)
- `p` Logical/scalar, If p=TRUE, results are in pourcentages, else, results are in counts.
- `case` String/scalar, Case of the letters taken into account ("all", "upper", "lower")
- `step` Integer/scalar, Step increment of positions when counting words.

Value

A matrix with nucleotide composition for all regions (N x M where N = 5*wordSize and M = length(from))

If error: NULL

Note

.Use setStrand by default

See Also

exactWord

Examples

```r
s<-'CGTACGTAGCTAGCTAGCTAGCTAGCTAGCTAGCTGATCGATGCTAGCTGATCGATGCT'
placeString(s)
x<-c(1,8,15,50)
y<-c(5,12,19,54)
compoSeq(from =x, to = y,wsize=2)
compoSeq(from =x, to = y,wsize=2,strand=1)
compoSeq(from =x, to = y,wsize=2,step=2,p=FALSE)
```

Concat

**Concatenation of two sequences**

Description

Concatenates a fragment of one sequence with a fragment of another one.

Usage

```r
concat(seqno1=0, seqno2=1, destSeqno=2, from1=1, to1=0, strand1=getStrand(), from2=1, to2=0, strand2=getStrand())
```
*deleteCR*

**Arguments**

- `seqno1` Integer/scalar, Integer/scalar, First sequence number (buffer number)
- `seqno2` Integer/integer, Integer/integer, Second sequence number (buffer number)
- `destSeqno` Integer/integer, Output sequence number (buffer number)
- `strand1` Integer/integer, Strand of the first fragment (forward: 0, reverse: 1)
- `from1,to1` Integer/integer, Absolute addresses of the beginning and the end of the first fragment, (1 means the first nucleotide and 0 conventionally the last one; from must not be larger than to and both vectors must be the same size).
- `from2,to2` Integer/integer, Absolute addresses of the beginning and the end of the second fragment, (1 means the first nucleotide and 0 conventionally the last one; from must not be larger than to and both vectors must be the same size)
- `strand2` Integer/integer, Strand of the second fragment (forward: 0, reverse: 1)

**Value**

- `destSeq` or -1 if error

**Author(s)**

L. Cottret

**See Also**

assemble, getSeq, appendSeq

**Examples**

```r
s1 <- "aaacgctagcgcg"
placeString(s1)
s2 <- "ttttctatcag"
placeString(s2,1)
concat(seqno1=0,seqno2=1, from1=2,to1=3, from2=8, to2=0, strand1=1)
getSeq(2)
# [1] "TTTCAG"
```

---

**deleteCR**

*Delete carriage return in file*

**Description**

Windows and Mac files do not use same terminating newline. This function convert file to Unix standard with only \n char for terminating newline.

**Usage**

deleteCR(file)
densityProfile

Arguments

file file name of a file

Value

0 if error, else 1.

Examples

seqNcbi("BY608190",file="BY608190.gbk",type="G")
deleteCR("BY608190.gbk")

densityProfile  Density profiles

Description

Computes profile(s) of user defined quantities around sites of interest in sequence fragments. Profile(s) is(are) constituted of bins of equal size around the sites of interest named origins. It produces for each bin, and for each quantity the mean, the standard deviation and the number of valid events.

Usage

densityProfile(ori, from, to, seqno = 0, fun = seqSkew,
fileout = NULL, nbinsL, nbinsR, sizeBin,
threshold=0,strand=getStrand(),accu=FALSE,case="all")
plot.profile (x,ylim=NULL,...)

Arguments

ori Integer/vector, Absolute address of the origins in each fragment
from, to Integer/vector, Absolute addresses of the beginning and the end of each fragment, (1 means the first nucleotide and 0 conventionally the last one; from must not be larger than to and both vectors must be the same size)
seqno Integer/scalar, Strand (forward: 0, reverse: 1)
fun Function, function to be used (by example seqSkew
fileout String/scalar, if not NULL, a file name to write results
nbinsL Integer/scalar, number of bins to create before the origin
nbinsR Integer/scalar, number of bins to create after the origin
sizeBin Integer/scalar, size of the bins
threshold Integer/scalar, For each bin, maximum number of N tolerated in the sequence to participate to the computation
strand Flag
accu Flag, if true, returns sum , sum of square, and count on demanding function; else returns, mean and standard error on mean.
case String/scalar, Case of the letters taken into account ("all", "upper", "lower")
densityProfile

x An element of class profile
ylim Range of y axis limits
... Graphical parameters can be given as arguments to `plot`.

Value

a list of matrices, with the mean(s), the standard deviation(s) and the number of valid sub-fragments in each bin.

Author(s)

Emma Marrakchi and Antoine Lucas

See Also

bankDensityProfile, GCcontent, seqSkew

Examples

s <- ""
for(i in 1:10)
  s <- paste(s, randomSeq(n=100), randomSeq(prob=c(0.3, 1, 1, 1, 0)/3, n=100), sep="")
placeString(s, seqno=0)

dens <- densityProfile(ori=200*(1:10)-100, from=200*(0:9)+1, to=200*(1:10),
  seqno=0, fun=seqSkew, nbinL=10, nbinR=10, sizeBin=10)
plot(dens$skta, main="TA skew")

## Example with flagged 'N'

## We create a sequence with a bias every 100 nucleotides
s <- ""
for(i in 1:10)
  s <- paste(s, randomSeq(n=100), randomSeq(prob=c(0.3, 1, 1, 1, 0.2)/3.5, n=100), sep="")
placeString(s, seqno=1)

dens2 <- densityProfile(ori=200*(1:10)-100, from=200*(0:9)+1, to=200*(1:10),
  seqno=1, fun=compoSeq, nbinL=10, nbinR=10, sizeBin=10)
plot(dens2$T, main="#T")

## The same but more permissive (allow 4 N in each bin)

dens3 <- densityProfile(ori=200*(1:10)-100, from=200*(0:9)+1, to=200*(1:10),
  seqno=1, fun=compoSeq, nbinL=10, nbinR=10, sizeBin=10, threshold=4)
plot(dens3$T, main="#T")

## Show numbers

dens
dens2
Description

Transformation from T to U in a sequence (dnaToRna), and vice versa (rnaToDna).

Usage

dnaToRna(seqno=0, from=1, to=0)
rnaToDna(seqno=0, from=1, to=0)

Arguments

seqno Integer/scalar, Sequence number (buffer number)
from, to Integer/scalar, Absolute addresses of the beginning and the end of the fragment, (1 means the first nucleotide and 0 conventionally the last one; from must not be larger than to and both vectors must be the same size)

Value

seqno or -1 if error

Author(s)

L. Cottret

Examples

s <- "atuuutututu"
placeString(s)
dnaToRna()
getSeq()
# [1] "auuuuuuuuuu"
rnaToDna()
getSeq()
# [1] "atttttttttt"
**exactWord**  

*Exact matches of an oligomer*

**Description**

Gets match positions of an oligomer in fragments of a sequence

**Usage**

```r
exactWord(word, seqno=0, from=1, to=0, strand = getStrand(), step=1, overlap=TRUE, wNbOcc =-1, case.sensitive=FALSE)
```

**Arguments**

- **word** string/scalar, Oligomer sequence
- **seqno** Integer/scalar, Sequence number (buffer number)
- **from, to** Integer/scalar, Absolute addresses of the beginning and the end of the fragment, (1 means the first nucleotide and 0 conventionally the last one; from must not be larger than to and both vectors must be the same size)
- **step** Integer/scalar, Size of the step in which search progression is done
- **overlap** TRUE: look for overlapping oligomer
- **wNbOcc** Integer/scalar, Maximum number of occurrences to retrieve sequentially in each fragment (-1 -> all)
- **strand** Integer/scalar, Strand (forward: 0, reverse: 1)
- **case.sensitive** T -> difference between lower and upper case letter, F -> no difference

**Value**

A list of match positions in each fragment. If error : NULL.

**Note**

step with negative values will be implemented soon

**Author(s)**

L.Cottret

**See Also**

`getSeq`
Examples

```r
s <- "cgtagctagctagctagctagctagctagcta"
placeString(s)
exactWord(word="ag", from=c(3,11,23), to=c(9,17,29))
# [[1]]
# [1]  4  8
#
# [[2]]
# [1] 12 16
#
# [[3]]
# [1] 24 28

placeString("TTTTTTTTTTTT")
exactWord("TTT")
exactWord("TTT", overlap=FALSE, step=2)
```

---

**fastaDescription**  
*Description field reading of a fasta sequence*

**Description**

Reads the description field of a fasta sequence

**Usage**

```r
fastaDescription(file, name="")
```

**Arguments**

- **file**  
  String/scalar, File name of the fasta sequences bank

- **name**  
  String/scalar, Name of the sequence whose description must be retrieved, (if not specified : first sequence of the file)

**Value**

The description field (the remaining of the heading line after the first space). if error : NULL

**Author(s)**

A. Lucas

**Examples**

```r
seqNcbi("BY608190", file="BY608190.fa")

fastaDescription(file="BY608190.fa", name="gi|26943372|gb|BY608190.1|BY608190")
```
Buffers

Low level functions on buffer manipulation

Description

nSeq returns the limit of buffers to use, or set this limit to maxBuffers when specified.
freeSeq free sequence seqno and complementary. Free all sequences for freeAllSeq

Usage

freeSeq(seqno=0)
freeAllSeq()
nSeq(maxBuffers=NULL)

Arguments

seqno  Integer, sequence number to free. (bufseq)
maxBuffers  Integer/Scalar, number of buffers

Value

seqno; 0 for freeAllSeq. Number of sequences for nSeq

Author(s)

A. Viari, L Cotteret, A Lucas

Examples

placeString("ATGAGTGATGAGATGATGAG",seqno=0)
placeString("ATGAGTGATGAGATGATGAG",seqno=3)
placeString("ATGAGTGATGAGATGATGAG",seqno=4)
revComp()
revComp(seqno=3)

sizeSeq()
## show size used in all buffers
.seqSize()

## freeSeq(3)
## free buffer 3
.seqSize()

freeAllSeq()
## free all buffers
.seqSize()

## show number of available buffers:
nSeq()
# 100
getAccn

Reading of a GeneR global variable

Description

Each of these functions returns GeneR global variable associated to a sequence buffer (see globals), but the global variable strand which concerns all the sequence buffers.

Usage

getAccn(seqno=0)
getBegSeq(seqno=0)
getSeqSize(seqno=0)
getStrand()
getEndSeq(seqno=0)

Arguments

seqno Integer/scalar, Sequence number (buffer number)

Details

All details on addresses and GeneR global variables can be found on page globals.

Value

getAccno Accession number or name of the sequence
getBegSeq,getEndSeq,getSeqSize
Beginning or ending position or size of the loaded sequence
getStrand Running working strand (0 -> forward, 1 -> reverse)
if error : -1

Author(s)

L. Cottret

See Also

setAccn,getParam,setParameter, globals
getParam

Reading of all the GeneR global variables

Description

Gives all the GeneR global variables associated to a sequence

Usage

ggetParam(seqno=0)

Arguments

seqno Integer/scalar, Sequence number (buffer number)

Details

All details on addresses and global variables can be found on page globals.

Value

A list with

Strand Running working strand (0 -> forward, 1 -> reverse)
begin beginning position of the loaded sequence
end ending position of the loaded sequence
size size of the parent sequence (to allow computations of relative adresses)
name name of the loaded sequence

Author(s)

L. Cottret

See Also

setParam

ggetSeq

Sequence fragments extraction

Description

Extraction of sequence fragments. ggetSeq further converts the fragments in character strings from the GeneR sequence buffer.

Usage

ggetSeq(seqno=0, strand = getStrand(), from=1, to=0)
Arguments

seqno  Integer/scalar, Sequence number (buffer number)
from, to  Integer/scalar, Absolute addresses of the begin and the end of the fragment, (1 means the first nucleotide and 0 conventionally the last one; from must not be larger than to and both vectors must be the same size)
strand  Integer/scalar, Strand (forward: 0, reverse: 1)

Value

A vector of character strings. if error : NULL

Author(s)

L.Cottret

See Also

assemble, concat, appendSeq, and for character string manipulation: substr

Examples

s<-'cgtagtagctagctagctagctagctagctag'
placeString (s, seqno=1)
getSeq(1,from=c(1,5,10),to=c(5,10,15))
# [1] "CGTAG" "GTAGCT" "TAGCTA"

# And on the reverse:
setStrand(1)
getSeq(1,from=c(1,5,10),to=c(5,10,15))

## The reverse complement
getSeq(1,strand=1)

indexFasta  

Index file creation for a sequences bank

Description

These functions create an index file for retrieving quickly a sequence into a fasta, genbank or embl sequence bank file.

Usage

indexFasta(file)
indexEmbl(file,index="ix")
indexGbk(file)

Arguments

file  String/scalar, File path and name of the sequences file
index  String/scalar, Suffix for the index file
Value

1 Indexfile has been created
-1 Error

Note

These functions create an index file even if it already exists.
Access number larger than 40 characters are skipped (a warning is returned). This can be increasing
with variable MAX_LEN_ACCNO in file GeneRGlobals.h (and rebuild GeneR library).

Author(s)

Antoine Lucas

See Also

makeIndex

Description

Inserts a sequence into another sequence

Usage

insertSeq(s, insert, from = 0)

Arguments

s String/scalar, The mother sequence
insert String/scalar, The sequence to be inserted
from Integer/scalar, Position of the insertion, 0 = last position, 1 = first

Details

Parameters insert and s are character string sequences and not GeneR sequences

Author(s)

Antoine Lucas

Examples

s<="gtcatgcgtctaggtcagtca"
insertSeq(s,"aaaaaaaaaaaaaaaa",7)
lowerSeq  

**Convert upper/lower-case characters in sequence fragments**

**Description**

Converts sequence fragments in lower or upper case letters

**Usage**

lowerSeq(seqno=0, from=1, to=0)

upperSeq(seqno=0, from=1, to=0)

**Arguments**

- **seqno**  
  Integer/scalar, Sequence number (buffer number)

- **from, to**  
  Integer/scalar, Absolute addresses of the begin and the end of the fragment, (1 means the first nucleotide and 0 conventionally the last one; from must not be larger than to and both vectors must be the same size)

**Author(s)**

Antoine Lucas

**See Also**

posMaskedSeq

**Examples**

```r
s<"aaaacgtagctagctacccccctagctacgtagattttt"
placeString(s, upper=FALSE)
x<1,21,39
y<4,28,0

upperSeq(from=x, to=y)
getSeq()
```

---

makeIndex  

**Index file creation for a bank file (internal function)**

**Description**

Makes an index for a bank file (if there wasn’t).

**Usage**

makeIndex(file, type="E", index="ix")
**Arguments**

- **file**: string/scalar, file name of bank file
- **type**: Sequence format ("E" for Embl, "G", for Genebank, "F" for Fasta)
- **index**: Suffix for the index file (usually: ix)

**Details**

Checks if index file exists and is newer than bank file. If not, calls one of indexFasta, indexEmbl, indexGbk functions.

**Value**

- 1: Index already exists (and no changes)
- 0: Index successfully created
- -1: Error

**Note**

Index files are in the form:

```
Accno   deb_feature  deb_sequence length_sequence
```

with one line by sequence. Number of char must be the same for each line (it is used to search a specific access number) but size used for accno is of 40 char by default. This can be change by setting variable `MAX_LEN_ACCNO` in GeneR_globals.h file and recompile the library.

**Author(s)**

Antoine Lucas

**See Also**

`indexFasta`, `indexEmbl`, `indexGbk`

**Examples**

```
seqNcbi("BY608190",file="BY608190.fa")

# Write index file BY608190.fa.ix ...
indexFasta("BY608190.fa")
```
**Description**

Mask regions delimited by from and to on a sequence buffer. This function delete the reverse complement if exists (should be recomputed).

**Usage**

```r
mask(seqno = 0, from = 1, to = 0, letter = "N")
```

**Arguments**

- `seqno`: Integer/scalaar, Sequence number (buffer number)
- `from, to`: Integer/scalaar, Absolute addresses of the beginning and the end of the fragment, (1 means the first nucleotide and 0 conventionally the last one; from must not be larger than to and both vectors must be the same size)
- `letter`: Letter to use for masking

**Value**

usually 1; NULL if error.

**Author(s)**

Antoine Lucas

**See Also**

See also `posMaskedSeq, upperSeq, lowerSeq`

**Examples**

```r
s <- "ATGCTgTGTTagctacAATGAGTGAGAGATGTGGGTTTaAAAattt"
placeString(s, upper=FALSE, seqno=0)
mask(from=c(10,20), to=c(15,22))
getSeq()
```
**Value Matching**

**Description**

'Match' returns a vector of the positions of (first) matches of its first argument in its second. Second must be an element of class segSet (or a numeric matrix with 2 columns) ordered and without overlapping segments (function or.segSet is designed for this purpose).

**Usage**

Match(x, a)

**Arguments**

x a vector of positions  
a an element of class segSet

**Value**

a vector of same size a x. Values are indices corresponding to element 'a', 0 when no segment found.

**Note**

This can be used for a texture, when we know only regions, described by set of segments.

**Author(s)**

Antoine Lucas

**Examples**

```r
a = matrix(c(1,30,40,50,60,70,80,110),ncol=2,byrow=TRUE)  
a = or.segSet(a)

## show a:

a

Match(1:40,a)

## texture sample:

b = matrix(c(15,18, 28,45,  
1,10, 15,20, 25,40,  
17,23, 35,38,100,105,  
110,120),ncol=2,byrow=TRUE)  
b = or.segSet(b)

texture = ( Match(1:120,a)>0 ) + ( Match(1:120,b)>0 )*2  
## change numbers to colors

texture <- as.factor(texture)
```
levels(texture) <- c("red","blue","green","yellow")
texture <- as.character(texture)
plot(1:120,rep(1,120),col=as.character(texture))

---

**not.globalSeg**

**Substraction of globals segments**

**Description**

Compute substraction of two objects of class globalSeg a and b, i.e. return segments from a that or
not in b.

When used with only one parameter, not(A) returns the complementary of each elements of A.

**Usage**

not.globalSeg(a, b = NULL)

**Arguments**

a,b  elements of class globalSeg

**Value**

An element of class globalSeg

**Author(s)**

Odile Rogier

**See Also**

globalSeg,and.globalSeg,not.segSet

**Examples**

```r
a = list(
  matrix( c( 1, 15, 17, 5, 45, 38),ncol=2),
  matrix( c(100, 120),ncol=2),
  matrix( c(130, 135, 140, 145),ncol=2),
  matrix( c(142, 160),ncol=2))

b = list(
  matrix( c(15, 28, 18, 45),ncol=2),
  matrix( c(1, 15, 25, 10, 20, 40),ncol=2),
  matrix( c(17, 35, 23, 38),ncol=2),
  matrix( c(100, 110, 105, 120),ncol=2))

a = as.globalSeg(a)
b = as.globalSeg(b)
```
not.segSet

Substraction of segments sets

Description
This function compute substraction of two objects of class segSet a and b, i.e. return segments from a that or not in b.

Usage
not.segSet(a, b)

Arguments
a, b elements of class segSet

Value
an element of class segSet.

Author(s)
Antoine Lucas

See Also
globalSeg, not.globalSeg

Examples

a = matrix(c(1,5,15,45,17,38,
100,120,130,140,
135,145,142,160),
ncol=2,byrow=TRUE)
b = matrix(c(15,18, 28,45,
1,10, 15,20, 25,40,
17,23, 35,38,100,105,
110,120),ncol=2,byrow=TRUE)
or.globalSeg

Union of global segments

Description

This function computes the union of two objects of class globalSeg a and b. When used with only one parameter, or(A) returns the union of A.

Usage

or.globalSeg(a, b = NULL, byrows = FALSE, ...)

Arguments

a, b elements of class globalSeg
byrows if TRUE, a and b must be lists of same length, and intersection is computed for each elements from a with its homologue in b.
... Unused

Value

An element of class globalSeg

Author(s)

Odile Rogier
or.segSet

See Also
globalSeg, and.globalSeg, or.segSet

Examples

```r
a = list(
  matrix( c( 1, 15, 17, 5, 45, 38),ncol=2),
  matrix( c(100 , 120),ncol=2),
  matrix( c(130, 135, 140, 145),ncol=2),
  matrix( c( 142 , 160),ncol=2))

b = list(
  matrix( c(15, 28, 18, 45),ncol=2),
  matrix( c(1, 15, 25, 10, 20, 40),ncol=2),
  matrix( c(17, 35, 23, 38),ncol=2),
  matrix( c(100, 110, 105, 120),ncol=2))

a = as.globalSeg(a)
b = as.globalSeg(b)

c = or(a,b)
par(mfrow=c(4,1))
plot(a,xlim=c(1,160),main="A")
plot(b,xlim=c(1,160),main="B")
plot(c,xlim=c(1,160),main="or(A,B)")

plot(or(b),xlim=c(1,160),main="or(B)")

## Show all

c
or(b)
```

or.segSet

Union of 2 segments sets

Description

Makes union of 2 segments sets

Usage

```r
or.segSet(a, b = NULL, simplify = TRUE, ...)
unionSeg(a, b = NULL, simplify = TRUE, ...)
```

Arguments

```r
a, b
  Elements of class segSet
simplify
  TRUE or FALSE. If TRUE, segments are sorted, and duplicated are removed.
...  Unused
```
placeString

Value
an element of class segment.

Author(s)
Antoine Lucas

See Also
globalSeg, or.globalSeg

Examples
from <- c(100, 75, 1, 25, 150)
to <- c(110, 120, 30, 50, 170)
or(as.segSet(data.frame(from, to)))
## return:
## from, to
## 1, 50
## 75, 120
## 150, 170

or.segSet(data.frame(from, to), simplify=FALSE)

## Tip to compute intergenic region
## (imagine: from = genes$start, to = genes$stop

x <- or.segSet(data.frame(from, to))
start <- x[,1]
stop <- x[,2]
n <- length(start)
tergenes <- cbind(stop[1:n-1], start[2:n])
tergenes

---

placeString Sequence loading in a GeneR sequence buffer.

Description
Puts a character string into a GeneR sequence buffer.

Usage
placeString(s, seqno=0, upper=TRUE)

Arguments
- **s** Character string to put into a sequence buffer
- **seqno** Integer/scalar, Sequence number (buffer number)
- **upper** upper case conversion (TRUE : conversion of the whole sequence in upper case letters, FALSE : sequence is taken as such)
Value

seqno or -1 if error.

Author(s)

L.Cottret

See Also

getSeq

Examples

s<"cgtagtagctagctagctagctagctagctag"
placeString (s, seqno=0)
getSeq(seqno=0)

placeString (s, seqno=1, upper=FALSE)
getSeq(seqno=1)

plot.globalSeg  Plot an object of class globalSeg

Description

Draw rectangles on an axis to represent our segments.

Usage

plot.globalSeg(x, xlim = range(x, global = TRUE),
border = "darkblue", col = "darkblue", density = NULL, at = NULL,
tick = TRUE, xlabels = TRUE, ...)

Arguments

x  Object of class GlobalSeg
xlim  (optional) vector of two elements:
col  color(s) to fill or shade the rectangle(s) with. The default 'NULL', or also 'NA' do not fill, i.e., draw transparent rectangles, unless 'density' is specified.
border  color for rectangle border(s). Can also be 'FALSE' to suppress the border, or 'TRUE' in which case 'col' is used.
density  the density of shading lines, in lines per inch. The default value of 'NULL' means that no shading lines are drawn. A zero value of 'density' means no shading lines whereas negative values (and 'NA') suppress shading (and so allow color filling).
posMaskedSeq

Description

These functions return the position of masked fragments within a sequence. Masked fragments are identified by lower case letters or by specific characters (for example: 'N').

Usage

posMaskedSeq(seqno=0,from = 1, to = 0, max = 10000,type= "lower")
posMaskedSeqFile(file, name = NA, from = 1, to = 0, max = 10000)
Arguments

- **seqno**: Integer/scalar, Sequence number (buffer number)
- **from,to**: Integer/vector, Absolute addresses of the begin and the end of the fragments, (1 means the first nucleotide and 0 conventionally the last one; from must not be larger than to)
- **file**: String/scalar, Path and file name of the Fasta sequence bank
- **name**: Name of the sequence (if NA: the first sequence of the bank)
- **type**: type of masked nucleotide i.e. "lower" or any character like "N" (if lower, posMaskedSeq looks for lower case letters; if a character, posMaskedSeq looks for this character)
- **max**: Maximum number of fragments to retrieve (default: 10000)

Value

- a matrix of size n,2.
- gives a warning if there is more than max regions.
- return numeric(0) when no regions found.

Note

- This function returns by default only 10000 first regions (default parameter max).

Author(s)

- Odile Rogier and Antoine Lucas

See Also

- See also mask, upperSeq, lowerSeq

Examples

```R
## Make a dummy sequence
s <- "ATGCTgTGTTagtacATHNNNNNNNNNNNNNTGGGTTtAAaattt"
placeString(s, upper=FALSE, seqno=0)
posMaskedSeq(seqno=0, type="upper")
posMaskedSeq(seqno=0, type="lower")

## Not run:
posMaskedSeq(seqno=0, type="lower", max=2)
## End(Not run)
posMaskedSeq(seqno=0, type="N")

writeFasta(file="toto.fa")
posMaskedSeqFile("toto.fa")
```
randomSeq

Create random sequence

Description

Function `randomSeq` creates a random sequence from a distribution of nucleotides, of poly-nucleotides. A real composition of nucleotides can be use from function `compoSeq`, with param `p=TRUE`.

ShuffleSeq creates a sequence while assembling at random specific number of each nucleotides (or poly-nucleotides). These number of nucleotide can be provided by function `compoSeq`, with param `p=FALSE`: it is then a re-assemblage of all nucleotides (or tri-nucleotides, or poly-nucleotides) of a real sequence.

Usage

```r
randomSeq(prob = c(0.25, 0.25, 0.25, 0.25, 0), letters = c("T", "C", "A", "G", "N"), n )
shuffleSeq(count, letters=c("T","C","A","G","N"))
```

Arguments

- `prob`: A vector of probability weights for obtaining the elements of the vector being sampled or a result from `compoSeq` function (with option `p=TRUE`.
- `count`: A vector of number of repetitions for each letters (or bi-tri nucleotides, must be of same length as letters) or a result from `compoSeq` function (with option `p=FALSE`).
- `letters`: Letters (or bi-tri nucleotides) to be sampled
- `n`: Integer giving the number of items to choose.

Value

A character string (sequence) or NULL.

Author(s)

A. Lucas

See Also

- `compoSeq`

Examples

```r
## Set seed of your choice (not requiered)
set.seed(3)

#### ---- RANDOMSEQ ----
## Create a sequence of size 30, GC rich
randomSeq(prob = c(0.20, 0.30, 0.20, 0.30), letters = c("T", "C","A","G"), n = 30)
## [1] "CTGGAACCGAGGGGTTCATCCCCCCAGTGA"
```
## SHUFFLESEQ

### Create a sequence with 7 T, 3 C and A, and 4 G.
```
shuffleSeq(count=c(7,3,3,4,0),letters=c("T","C","A","G","N"))
```

### Same with bi-nucleotides
```
shuffleSeq(count=c(rep(4,4),rep(2,4),rep(1,4),rep(0,4)),letters = c("TT","TC","TA","TG","CT","CC","CA","CG","AT","AC","AA","AG","GT","GC","GA","GG"))
```

### From a real sequence:
```
seqNcbi("BY608190",file="BY608190.fa")
```
```
readFasta("BY608190.fa")
```

### create a random sequence from a real tri-nucleotides distribution
### Size of sequence will be 10*3.
```
randomSeq(compoSeq(wsize=3,p=TRUE),n=10)
```

### re assemble real tri-nucleotides of a real sequence
```
shuffleSeq(compoSeq(wsize=3,from=1,to=30,p=FALSE))
```

---

### readEmblDescript

#### Read features from an Embl bank file

**Description**

Read features from an Embl bank file

**Usage**

```
readEmblDescript(file, name = NA, code = "DE")
```

**Arguments**

- **file**: String/scalar, File name of the bank (example: "foo.embl")
- **name**: String/scalar, Sequence name (default: first sequence)
- **code**: String/scalar, feature code to look for (i.e. "CC", "DE", "CC", etc...)

**Value**

A vector of character string.

**Author(s)**

Antoine Lucas
## Not run:
download.file("http://bioinfo.hku.hk/sars/AY291451.seq",
destfile="AY291451.seq")

readEmblDescipt(file = "AY291451.seq",name ="AY291451",code ="DE")
##_returns: "SARS coronavirus TW1, complete genome."

readEmblDescipt(file = "AY291451.seq",name ="AY291451",code ="OC")
##_returns: "Viruses; ssRNA positive-strand viruses, no DNA stage; Nidovirales; Coronavirus; SARS coronavirus."

readEmblDescipt(file = "AY291451.seq",name ="AY291451",code ="RA")

## End(Not run)

---

### Read location

---

**Get annotations of a GeneBank or an EMBL sequence**

**Description**

Extracts by keywords, from a GeneBank or an EMBL sequence entry, annotations from the features field.

**Usage**

```r
readGbkgLocation (file, name = NA, from = 1, to = 0, key = "CDS", subkey = "")
readEmblLocation (file, name = NA, from = 1, to = 0, key = "CDS", subkey = "")
```

**Arguments**

- **file**: String/scalar, File name of the bank
- **name**: String/scalar, Sequence name (default: first sequence)
- **from, to**: Integer/scalar, Absolute addresses of the begin and the end of the fragment, (1 means the first nucleotide and 0 conventionally the last one; from must not be larger than to)
- **key**: String/scalar, Feature to retrieve. (ex: GENE, CDS...)
- **subkey**: String/scalar, Label to retrieve (ex: locus_tag, note, codon_start...).

**Value**

A data frame
readSeq

Sequence extraction from a bank

Description

These functions load a sequence from a Fasta, Embl or GeneBank sequence bank file into a GeneR sequence buffer or as a character string.

readSeq is the generic function, readFasta, readGbk and readEmbl are aliases.

Usage

readSeq(file, name=NA, type="F", seqno=0,
          from=1, to=0, upper=TRUE, index="ix")
readFasta(file, name=NA, seqno=0,
          from=1, to=0, upper = TRUE, index="ix")
strReadFasta (file, name = NA, from = 1,
    to = 0, upper = TRUE)
readGbk(file, name=NA, seqno=0, from=1, to=0, upper = TRUE, index = "ix")
strReadGbk (file, name = NA, from = 1,
    to = 0, upper = TRUE)
readEmbl(file, name=NA, seqno=0, from=1, to=0, upper = TRUE, index = "ix")
strReadEmbl (file, name = NA, from = 1,
    to = 0, upper = TRUE, index = "ix")

Arguments

file       String/scalar, File name of the bank
name       String/scalar, Sequence name (default: first sequence)
upper      upper case letters conversion (TRUE: The whole sequence is converted to upper
case letters, FALSE: no conversion (only implemented for Fasta sequence)).
type       String/scalar, Bank format ("F" -> fasta. "E" -> embl, "G" -> GenBank)
seqno      Integer/scalar, Sequence number (buffer number)
from, to   Integer/scalar, Absolute addresses of the begin and the end of the fragment, (1
           means the first nucleotide and 0 conventionally the last one; from must not be
           larger than to)
index      String/scalar, type of index ("ix" -> sequence name is the accession number
           "id" -> sequence name is the identifier)

Value

integer: 1 if OK; -1 if error. strReadFasta, strReadGbk and strReadEmbl returns the sequence as
character string.

Author(s)

L. Cottret, A. Lucas.

See Also

GeneR, globals, readEmblLocation, readGbkLocation, getSeq, sizeSeqEmbl, sizeSeqFasta

Examples

## Get a sequence from Ncbi
seqNcbi("BY608190", file="BY608190.gbk", type="G")

## Read Gbk file to buffer 0
readGbk("BY608190.gbk")

## Or to a character string
strReadGbk("BY608190.gbk")

## Write to Fasta file
writeFasta(file='toto.fa')

## Make an index to this file
indexFasta('toto.fa')
readFasta (file="toto.fa",seqno=0,from=1,to=159)

## Show sequence on buffer 0
getSeq()

## Make a Fake file
writeEmblLine(file='toto.embl',code='AC',header='tmp',append=FALSE)
writeEmblSeq(file='toto.embl')

## Make index on file toto.embl
indexEmbl('toto.embl')

## Read Embl file to buffer 0
readEmbl('toto.embl')

## Or read "directly"
strReadEmbl('toto.embl')

---

**relist**

*Group segments into global segments*

**Description**

`relist`, `relistage` : put all segments into its specific enveloppe.

**Usage**

`relist(ranges, envelop)`
`relistage(ranges, envelop)`

**Arguments**

`envelop, ranges` elements of class `segSet` or matrix `nx2`

**Value**

`relist` a vector with envelopes indices (-1 if none found)
`relistage` an element of class `globalSeg`

**Note**

Envelops must overlap ranges.

**Author(s)**

Antoine Lucas and Odile Rogier
Reverse complementary

Examples

```r
from <- c(1, 15, 17, 100, 130, 135, 142)
to <- c(40, 45, 38, 120, 140, 145, 160)
envelfrom <- c(1, 100, 130)
envelto <- c(45, 120, 160)
ranges <- as.segSet(data.frame(from, to))
envel <- as.segSet(data.frame(envelfrom, envelto))
relist(ranges, envel)
c <- relistage(ranges, envel)
```

```r
par(mfrow=c(3,1))
plot(ranges, xlim=c(1, 160), main="ranges")
plot(envel, xlim=c(1, 160), main="Envelopes")
plot(c, xlim=c(1, 160), main="relist")
```

Reverse complementary

*Performs the reverse of a sequence*

Description

These functions perform the reverse/complementary of a sequence. RevComp loads further the result into the transparent complementary buffer.

Usage

```r
revComp(seqno=0, force=FALSE)
strComp(s)
```

Arguments

- `seqno` Integer/scalar, Sequence number (buffer number)
- `force` Logical/scalar, Force flag (FALSE -> revComp creates the reverse/complementary sequence only if it doesn’t exist, TRUE -> revComp creates it anyway)
- `s` A sequence, string format.

Value

`seqno` or `-1` if error.

Author(s)

L. Cottret

See Also

`getSeq`
Examples

```r
s <- "cgtagtagctagctagctagctagctag"
placeString (s, seqno=1)
getSeq(1)
# return [1] "CGTAGTAGCTAGCTAGCTAGCTAGCTAG"

revComp(1)  # computes the reverse
getSeq(1,1)
# return [1] "CTAGCTAGCTAGCTAGCTAGCTACTACG"

# Or with strComp
strComp (s)
```

seqSkew

compute the strand assymetries from one sequence fragment

Description

compute the strand assymetries from one sequence fragment

Usage

```r
seqSkew(seqno = 0, from=1, to=0, strand=getStrand(), case = "all")
GCcontent(seqno = 0, from=1, to=0, case = "all", ...)
```

Arguments

- `seqno` Input sequence number. (bufseq)
- `from, to` Begining and ending of sequence, can be vectors. 0 represent the last nucleotide and 1 the first one.
- `case` "all"
- `strand` strand
- `...` other parameters

Value

a data frame with 4 columns as follow

- TA skew or TA pourcent
  \[(T-A)/(T+A) \text{ or } \text{count(TA)} / \text{count(TAGCN)}\]
- GC skew or GC pourcent
  \[(G-C)/(G+C) \text{ or } \text{count(GC)} / \text{count(TAGCN)}\]
- total skew or C pourcent
  \[\text{TA+GC skews or count(C) / count(TAGCN)}\]

Author(s)

Yves d’Aubenton and Emma
**seqSrs**

*Download a sequence in Fasta / Genbank / Embl format from Ncbi or Srs*

Get a sequence in Fasta / Genbank / Embl format through web with only an accno.

seqSrs will get the sequence through a srs web program. seqNcbi will do the same with ncbi web server (I prefer this one).

**Usage**

```r
seqSrs (accno, file="toto.seq", submotif=FALSE,
srs="http://srs.ebi.ac.uk/srsbin/cgi-bin/wgetz",
bank=c("EMBL","REFSEQ"))
seqNcbi (accno, file="toto.seq", submotif=FALSE, type="fasta")
```

**Arguments**

- **accno**: Access Number: "dbjBY608190.1|BY608190" or "BY608190"
- **file**: file where sequence will be downloaded
- **submotif**: a logical value indicating whether we look for a subpattern of accno
- **srs**: a url srs web program
- **type**: fasta or genbank for seqNcbi (file format of sequence).
- **bank**: List of banks to search into.

**Value**

1 if file has been correctly created. A file containing the sequence in file format requested

**Note**

SeqNcbi returns sometimes Genbank file in a not very valid format (specially with EST sequences). These files will not be computed by readseq. (Use Fasta format, then).
**SetAccn**

**Author(s)**

Antoine Lucas, Centre de Genetique moleculaire, CNRS Gif / Yvette

**Examples**

```r
seqNcbi("BY608190",file="BY608190.fa")

## Not run:
# idem:
seqSrs("dbj\|BY608190.1\|BY608190",file="BY608190.embl",submotif=TRUE,type="embl")
seqSrs("AK129232",type="embl",srs="http://www.infobiogen.fr/srs71bin/cgi-bin/wgetz")
## End(Not run)
```

---

**SetAccn**  
*Set globals variables of a sequence*

**Description**

Globals variables concerning a sequence (see **globals**) can be directly set with these functions. Global variable *strand* concerns all buffers of sequence.

**Usage**

```r
setAccn(name, seqno=0)
setBegSeq(pos=1, seqno=0)
setSeqSize(pos, seqno=0)
setStrand(str=0)
```

**Arguments**

- **name**  
  Character: name of sequence

- **seqno**  
  Integer: sequence number (bufno)

- **pos**  
  Integer: size, or position.

- **str**  
  Strand: 0 : watson (forward)  
  1 : crick (reverse)

**Details**

All details on addresses and globals variables are on page **globals**.

**Value**

The name, or "error" if error.

**Author(s)**

L. Cottret
SetParam

Set global variables associated to a sequence

Description

Set global variables associated to a sequence

Usage

setParam(from = 1, sizeMaster = seqSize(), seqno = 0)

Arguments

from    Integer, begin of sequence.
sizeMaster    Integer, size of origin sequence.
seqno    Integer, sequence number (bufseq)

Value

-1 if error or the list.

Author(s)

L. Cottret

See Also

globals, getAccn, setParam, getParam

size.globalSeg

Size.globalSeg

Description

Tools manipulating size,max,min,length of elements of class globalSeg

Usage

size.globalSeg(a)
Max.globalSeg(a)
Min.globalSeg(a)
Length.globalSeg(a, global=TRUE)
Arguments

a elements of class globalSeg

global if TRUE: return one value (number segments) if FALSE return a vector (number of segments in each elements of the list)

Value

return a vector with min, max, size or length of each element of the list (a element of class globalSeg is a list of elements of class segments).

Author(s)

Odile Rogier

Examples

a = list(
    matrix( c( 1, 15, 17, 5, 45, 38), ncol=2),
    matrix( c(100, 120), ncol=2),
    matrix( c(130, 135, 140, 145), ncol=2),
    matrix( c(142, 160), ncol=2))
a = as.globalSeg(a)

Length(a)
Length(a, global=TRUE)
size(a)
Max(a)
Min(a)

SizeSeqFasta

Size of a sequence from a fasta/embl/gbk file

Description

Give the size of a sequence from a fasta/embl/gbk file.

Usage

sizeSeqFasta(file, name=NA)
sizeSeqEmbl(file, name=NA, index="ix")
sizeSeqGbk(file, name=NA)

Arguments

file File name of the bank
name Name of the sequence in the bank (default: first sequence)
index character. "ix": Name is accn of sequence.
         "id": Le name is id of sequence.
SizeSeq

Size of a sequence in a buffer

Description

Get size (number of characters) of a sequence. `.seqSize` returns size allocated in all buffers.

Usage

```r
sizeSeq(seqno=0)
```

Arguments

- `seqno` Integer: sequence number (bufseq)

Value

Integer: size of sequence. A list of two vectors for `.seqSize`.

Examples

```r
# NCBI gbk format
seqNcbi("NC_004718", file="NC_004718.gbk", type="G")
sizeSeqGbk("NC_004718.gbk")
# Or:
sizeSeqGbk("NC_004718.gbk","NC_004718")

# Idem with fasta format
seqNcbi("NC_004718", file="NC_004718.fa", type="F")
sizeSeqFasta("NC_004718.fa")
# Or:
sizeSeqFasta("NC_004718.fa","gi|30271926|ref|NC_004718.3|")

## Not run:
# With Embl
download.file("http://bioinfo.hku.hk/sars/AY291451.seq",
destfile="AY291451.seq")
sizeSeqEmbl("AY291451.seq")
## End(Not run)
sliceSegment

Author(s)

L. Cottret

See Also

sizeSeqEmbl, sizeSeqFasta

Examples

freeAllSeq()
s<="cgtagtagctagctagctagctagctagctag"  
placeString (s, seqno=1)
sizeSeq(1)
.seqSize()

sliceSegment  

delineate sub-segments of equal size at both sides of an origin in a segment

Description

delineate sub-segments of equal size at both sides of an origin in a segment

Usage

sliceSegment(from = 1, to = 0, ori = 0, nbinL = 0, nbinR = 0, size = 1)

Arguments

from integer, position of the begin of segment
to integer, position of the end of segment
ori integer, the position of the origin
nbinL number of sub-segments to create before the origin
nbinR number of sub-segments to create after the origin
size integer, size of the sub-segments

Value

an element of class segSet

Author(s)

Yves d’Aubenton and Emna

See Also

bankDensityProfile, densityProfile, GCcontent, seqSkew
Examples

```r
## If by example we have a gene in positions 150 to 200
## This will create segments to study
## A more complex example is provided with function densityProfile
sliceSegment(from=1,to=200,ori=150,nbinL=15,nbinR=15,size=10)
```

---

## Translation from DNA trinucleotides to proteine

**Description**

Translation tools from DNA trinucleotides to proteine

**Usage**

```r
translate (seqno, from = 1, to = 0, strand = 0, code = 0,
charcode = "")
strTranslate (s, code = 0, charcode = "")
showTable (code = 0, charcode = ")
```

**Arguments**

- `segno`: Integer, sequence number (bufno)
- `s`: Sequence as a character string
- `from, to`: Beginning and ending of sequence, can be vectors. 0 represent the last nucleotide and 1 the first one.
- `strand`: 0: forward, 1: reverse, can be a vector
- `code`: One of the following standard code: 0, standard genetic code; 1 Vertebrate Mitochondrial Code; 2 Yeast Mitochondrial Code; 3 Mold, Protozoan, and Coelenterate Mitochondrial Code and the Mycoplasma/Spiroplasma Code; 4 Invertebrate Mitochondrial Code
- `charcode`: A character string of size 64, like "FFLLSSSSYY**CC*WLLLLPPPPHHHQQRRRRHHHHQQGGGG" for translation code to use, in the order: TTT TTC TTA TTG TCT TCC TCA ...(for F F L L S S S). Use showTable to be sure!

**Value**

- `strTranslate`: return a character string of the protein
- `translate`: return a vector of character string of the protein
- `showTable`: return the table of translation

All return -1 if error.

**Note**

Global value of *strand* has no effect on this function. (see `globals`, `getParam`, `setStrand`
Examples

s<-"gtcatgcatgtaggtaggttaaatgcgtctaggtgacagtctaacaa"

# Simple usage:
strTranslate(s)
#[1] "VMHAR*QLKCV*VTV*Q"

# with buffers
placeString(s)
translate()
# the same...
#[1] "VMHAR*QLKCV*VTV*Q"

# with 2 positions
translate (from=c(1,2),to=c(0,0))
#[1] "VMHAR*QLKCV*VTV*Q" "SCMLGDS*NASR*QSN"

# with 2 strands
translate (from=c(1,2),to=c(0,0,0),strand=c(0,0,1))
#[1] "VMHAR*QLKCV*VTV*Q" "SCMLGDS*NASR*QSN" "LLDCHLDAF*LSPSMHD"

# With Invertebrate Mitochondrial Code
translate(code=4)
#[1] "VMHASWQLKCV*VTV*Q"

# With a personal code
translate(charcode="FFLLxxxxYY**CCwwLLLLPPPPHHQQRRRIIMMTTTTNKKSSSRuuuuAAAAADDEEGGGG")
#[1] "uMHARwQLKCu*uTu*Q"

# Show what is this code...
showTable(charcode="FFLLxxxxYY**CCwwLLLLPPPPHHQQRRRIIMMTTTTNKKSSSRuuuuAAAAADDEEGGGG")
# [,1] [,2]
#[1,] "UUU" "F"
#[2,] "UUC" "F"
#[3,] "UUA" "L"
#[4,] "UUG" "L"
#[5,] "UCU" "x"
#[6,] "UCC" "x"
#[7,] "UCA" "x"
... 

# Show Standard table:
showTable()

# Show Invertebrate Mitochondrial Code
showTable(code=4)

WriteEmblSeq  Write to Embl file

Description

Write a sequence, with description into a EMBL file. writeEmblSeq write the sequence; writeEmblLine write a feature line, writeEmblComment write a comment line. CompleteStringWithSpace:
internal function

Usage

writeEmblSeq (file, seqno = 0)
writeEmblLine (file, code = "", header = "", text = "", nextfield = TRUE, append=TRUE)
writeEmblComment(file, code = "", text = "", nextfield = TRUE, append=TRUE)

Arguments

file EMBL File name
code 2 letters: to be written at line beginning
header First part of line
text a text to be written
seqno Integer, sequence number (bufno)
nextfield TRUE: write XX after line, FALSE: don’t write XX.
append Scalar boolean. TRUE -> line(s) is added at the end of the file. FALSE -> file is written over.

Author(s)

A. Lucas and Vincent Lefort

Examples

s<-"gtcatgcatgcatggacagttaaatgcgtctaggtgacagtctaacaa"
placeString(s)

# Add lines with "CC bla bla bla" and a line "XX"
writeEmblComment(file="toto.embl",code="CC",text="This is a comment for \
this dummy sequence... I try to be long enough to show that this comment \will be written on several lines",append=FALSE)

# Add a line with "FT CDS bla bla bla"
writeEmblLine(file="toto.embl",code="FT",header="CDS",text="<1..12",
nextfield = FALSE)
# Add lines with "FT bla bla bla"
writeEmblLine(file="toto.embl",code="FT",header="",text="/codon_start=2",
nextfield = FALSE)
writeEmblLine(file="toto.embl",code="FT",header="",text="/gene="toto"",
nextfield = FALSE)
writeEmblLine(file="toto.embl",code="FT",header="",text="/note="Here is \what I think about this gene\"",nextfield = FALSE)

### Translation
prot <- translate(seqno=0,from=getOrfs()[1,1],to=getOrfs()[1,2])
writeEmblLine (file="toto.embl",code='FT',header='',
text=paste('/translation="',prot ,"",sep=''),nextfield =TRUE)

# Add sequence
writeEmblSeq(file="toto.embl")

## Show file
cat(paste(readLines("toto.embl"), collapse='\n'))

---

**WriteFasta**  
*Write sequences into fasta file format*

**Description**

Write one or more parts of sequences into Fasta file format.

**Usage**

```r
writeFasta(file ="data.fasta", seqno = 0, from = getBegSeq(seqno),
 to = getEndSeq(seqno), name = getAccn(seqno),
 comment = paste("from", from, "to", to),
 cpl = 60, append = FALSE)
```

```r
strWriteFasta(s, file ="data.fasta", name = "Seq_R",
 comment = as.character(nchar(s)),
 cpl = 60, append = FALSE)
```

**Arguments**

- `seqno`  
  Integer, number of the sequence. (bufseq)

- `s`  
  A character string containing the sequence

- `from, to`  
  Begining and ending of sub-sequences to extract. Can be vectors. 0 represent the last nucleotide and 1 the first one.

- `name`  
  Sequence name.

- `comment`  
  Comment.

- `file`  
  Output File.

- `cpl`  
  Number of caracters by line.

- `append`  
  Scalar boolean. TRUE -> sequence is added at the end of the file. FALSE -> file is written over.

**Value**

1 or -1 if error.

**Author(s)**

L. Cottret
Examples

```r
s<="cgtagctagctagctagctagctacgtagctagctgactgtcgat"
placeString(s)
from <- c(1,14)
to <- c(10,0)
writeFasta(file="bank.fasta",from =from, to=to, append=FALSE)

strWriteFasta(s="ATGTCGTGGTaaattaatTTGGTCCC",file="bank.fasta",append=TRUE)
```

```r
## Show file
cat(paste(readLines("bank.fasta"),collapse='\n'))
```

---

**Xor.globalSeg**  
*Xor for global segments*

**Description**

computes the eXclusive OR of two objects of class globalSeg `a` and `b`, i.e. returns segments which correspond to at least one part of a segment in one set but to nothing in the other set.

When used with only one parameter, Xor(`a`) returns segments belonging to only one input segment.

see the example for a more comprehensive visualisation.

**Usage**

```r
Xor.globalSeg(a, b = NULL,...)
```

**Arguments**

- `a, b` elements of class globalSeg
- `...` Unused

**Value**

An element of class globalSeg

**Author(s)**

Odile Rogier

**See Also**

`globalSeg, and.globalSeg, Xor.segSet`
Xor.segSet

Examples

```r
a = list(
    matrix(c(1, 15, 17, 5, 45, 38), ncol=2),
    matrix(c(100, 120), ncol=2),
    matrix(c(130, 135, 140, 145), ncol=2),
    matrix(c(142, 160), ncol=2))

b = list(
    matrix(c(15, 28, 18, 45), ncol=2),
    matrix(c(1, 15, 25, 10, 20, 40), ncol=2),
    matrix(c(17, 35, 23, 38), ncol=2),
    matrix(c(100, 110, 105, 120), ncol=2))

c = Xor(a, b)
b = as.globalSeg(b)
c = as.globalSeg(a)
```

Xor.segSet

**Xor for segments sets**

Description

computes the eXclusive OR of two objects of class segSet `a` and `b`, i.e., returns segments which correspond to at least one part of a segment in one set but to nothing in the other set.

XorRecouvr returns segments which correspond to at least one part of a segment of the envelope but to nothing in the segment set

Usage

```r
Xor.segSet(a, b)
xorRecouvr(ranges, env)
```

Arguments

- `a, b`: elements of class segSet, or matrix nx2
- `ranges, env`: elements of class segSet, or matrices nx2

Value

an element of class segSet.
Author(s)
Antoine Lucas

See Also
  globalSeg, not.globalSeg

Examples

```r
a = matrix(c(1,5, 15,45,17,38,
100,120,130,140,
135,145,142,160),
ncol=2,byrow=TRUE)
b = matrix(c(15,18, 28,45,
1,10, 15,20, 25,40,
17,23, 35,38,100,105,
110,120),ncol=2,byrow=TRUE)
a <- as.segSet(a)
b <- as.segSet(b)
c = Xor(a,b)
par(mfrow=c(3,1))
plot(a,xlim=c(1,160),main="A")
plot(b,xlim=c(1,160),main="B")
plot(c,xlim=c(1,160),main="A Xor B")
```

```r
## Another sample
a = matrix(c(1,30,40,50,60,70,80,110),ncol=2,byrow=TRUE)
b = matrix(c(1,10,20,30,40,70,80,90,100,110),ncol=2,byrow=TRUE)
a <- as.segSet(a)
b <- as.segSet(b)
c = Xor(a,b)
par(mfrow=c(3,1))
plot(a,xlim=c(1,160),main="A")
plot(b,xlim=c(1,160),main="B")
plot(c,xlim=c(1,160),main="A Xor B")
```

```r
## Show all
  c
```
Index

*Topic database
  - Read location, 42
  - readEmblDescript, 41

*Topic utilities
  - and.globalSeg, 6
  - and.segSet, 8
  - appendSeq, 9
  - as.globalSeg, 10
  - as.segSet, 10
  - assemble, 11
  - AtoR, 12
  - bankDensityProfile, 13
  - bankSummary, 15
  - Buffers, 23
  - CompoSeq, 16
  - Concat, 17
  - deleteCR, 18
  - densityProfile, 18
  - DnaToRna, 20
  - exactWord, 21
  - fastaDescription, 22
  - GeneR, 1
  - getAccn, 24
  - getOrfs, 5
  - getParam, 25
  - getSeq, 26
  - Globals Variables, 3
  - globalSeg, 2
  - indexFasta, 27
  - insertSeq, 27
  - lowerSeq, 28
  - makeIndex, 29
  - mask, 30
  - Match, 31
  - not.globalSeg, 32
  - not.segSet, 33
  - or.globalSeg, 34
  - or.segSet, 35
  - placeString, 36
  - plot.globalSeg, 37
  - posMaskedSeq, 38
  - randomSeq, 40
  - Read location, 42
  - readEmblDescript, 41
  - readSeq, 43
  - relist, 45
  - Reverse complementary, 46
  - seqSkew, 47
  - seqSrs, 48
  - SetAccn, 49
  - SetParam, 50
  - size.globalSeg, 50
  - SizeSeq, 52
  - sizeSeqFasta, 51
  - sliceSegment, 53
  - Translate, 54
  - WriteEmblSeq, 55
  - WriteFasta, 57
  - Xor.globalSeg, 58
  - Xor.segSet, 59
  - -.globalSeg(not.globalSeg), 32
  - -.segSet(not.segSet), 33
  - .seqSize(SizeSeq), 52
  - &.globalSeg(and.globalSeg), 6
  - |.globalSeg(or.globalSeg), 34
  - and(and.segSet), 8
  - and.globalSeg, 6, 8, 32, 35, 58
  - and.segSet, 7, 8
  - appendSeq, 9, 12, 17, 26
  - as.data.frame.segSet(as.segSet), 10
  - as.globalSeg, 3, 10
  - as.matrix.globalSeg
  - (as.globalSeg), 10
  - as.matrix.segSet(as.segSet), 10
  - as.segSet, 3, 10
  - assemble, 9, 11, 17, 26
  - AtoR, 4, 12
  - AtoT, 4, 13
  - AtoT(AtoR), 12
  - bankDensityProfile, 13, 15, 19, 48, 53
  - bankSummary, 14, 15
  - Buffers, 23
  - CompoSeq, 16
INDEX

setStrand(SetAccn), 49
showTable(Translate), 54
shuffleSeq(randomSeq), 40
size(size.globalSeg), 50
sizeSeq, 52
sizeSeqFasta, 44, 53
sizeSeqEmbl(SizeSeqFasta), 51
sizeSeqFasta, 51
sizeSeqFastaFasta, 44, 53
sizeSeqFasta(SeqSeqFasta), 51
sizeSeqFastaGbk(SeqSeqFasta), 51
sliceSegment, 53
strComp, 3
strComp(Reverse complementary), 46
strCompSeq(CompoSeq), 16
strReadEmbl(readSeg), 43
strReadFasta, 3
strReadFasta(readSeg), 43
strReadGbk(readSeg), 43
strTranslate(Translate), 54
strWriteFasta(WriteFasta), 57
substr, 26

Translate, 54
translate(Translate), 54
TtoA, 4, 13
TtoA(AtoR), 12
TtoR, 4, 13
TtoR(AtoR), 12

unionSeg, 3
unionSeg(or.segSet), 35
upperSeq, 30, 39
upperSeq(lowerSeq), 28

writeEmbl(WriteEmblSeq), 55
writeEmblComment(WriteEmblSeq), 55
writeEmblLine(WriteEmblSeq), 55
WriteEmblSeq, 55
writeEmblSeq(WriteEmblSeq), 55
WriteFasta, 57
writeFasta(WriteFasta), 57

Xor(Xor.segSet), 59
Xor.globalSeg, 58
Xor.segSet, 58, 59
xorRecouv(Xor.segSet), 59