Package 'Modstrings'

February 28, 2026

Type Package

Title Working with modified nucleotide sequences

Version 1.26.0

Date 2022-08-13

Description Representing nucleotide modifications in a nucleotide sequence
is usually done via special characters from a number of sources. This
represents a challenge to work with in R and the Biostrings package.
The Modstrings package implements this functionallity for RNA and DNA
sequences containing modified nucleotides by translating the character
internally in order to work with the infrastructure of the Biostrings
package.
For this the ModRNAString and ModDNAString classes and derivates and
functions to construct and modify these objects despite the encoding issues
are implemenented. In addition the conversion from sequences to list like
location information (and the reverse operation) is implemented as well.

License Artistic-2.0

Encoding UTF-8

biocViews DataImport, DataRepresentation, Infrastructure, Sequencing,
Software

Depends R (>= 3.6), Biostrings (>= 2.51.5)

Imports methods, BiocGenerics, GenomicRanges, S4Vectors, IRanges,
XVector, stringi, stringr, crayon, grDevices

Suggests BiocStyle, knitr, rmarkdown, testthat, usethis

Collate 'Modstrings.R' 'AllGenerics.R'
'Modstrings-external-functions.R'
'Modstrings-external-C-calls.R' 'Modstrings-ModStringCodec.R'
'Modstrings-ModString.R' 'Modstrings-ModStringSet.R'
'Modstrings-ModStringViews.R' 'Modstrings-MaskedModString.R'
'Modstrings-ModStringCodec-data.R'
'Modstrings-ModStringSet-io.R' 'Modstrings-ModStringSetList.R'
'Modstrings-QualityScaledModStringSet.R'
'Modstrings-letterFrequency.R' 'Modstrings-modifyNucleotide.R'
'Modstrings-replaceLetterAt.R' 'Modstrings-sanitize.R'
'Modstrings-separate.R' 'Modstrings-seqtype.R' 'datasets.R'
'utils.R' 'zzz.R'

VignetteBuilder knitr

RoxygenNote 7.3.3

BugReports <https://github.com/FelixErnst/Modstrings/issues>

git_url https://git.bioconductor.org/packages/Modstrings

git_branch RELEASE_3_22

git_last_commit a4e48ba

git_last_commit_date 2025-10-29

Repository Bioconductor 3.22

Date/Publication 2026-02-27

Author Felix G.M. Ernst [aut, cre] (ORCID:
     <<https://orcid.org/0000-0001-5064-0928>>),
    Denis L.J. Lafontaine [ctb, fnd]

Maintainer Felix G.M. Ernst <felix.gm.ernst@outlook.com>

Contents

Modstrings-package   *Modstrings: Working with modified nucleotide sequences*

Description

Representing nucleotide modifications in a nucleotide sequence is usually done via special characters from a number of sources. This represents a challenge to work with in R and the Biostrings package. The Modstrings package implements this functionallity for RNA and DNA sequences containing modified nucleotides by translating the character internally in order to work with the infrastructure of the Biostrings package. For this the ModRNAString and ModDNAString classes and derivates and functions to construct and modify these objects despite the encoding issues are implemenented. In addition the conversion from sequences to list like location information (and the reverse operation) is implemented as well.

Author(s)

Maintainer: Felix G.M. Ernst <felix.gm.ernst@outlook.com> (ORCID)

Other contributors:

- Denis L.J. Lafontaine <denis.lafontaine@ulb.ac.be> [contributor, funder]

See Also

Useful links:

- Report bugs at https://github.com/FelixErnst/Modstrings/issues

---

letterFrequency	*Calculate the frequency of letters in nucleotide sequence with modifications, or the consensus matrix of a set of sequences*

Description

These functions follow the same principle as the Biostrings functions. Please be aware, that the matices can become quite large, since the alphabet of ModString objects contains more letters.

Usage

```
## S4 method for signature 'ModDNAString'
hasOnlyBaseLetters(x)

## S4 method for signature 'ModRNAString'
hasOnlyBaseLetters(x)

## S4 method for signature 'ModDNAString'
alphabetFrequency(x, as.prob = FALSE, baseOnly = FALSE)

## S4 method for signature 'ModRNAString'
alphabetFrequency(x, as.prob = FALSE, baseOnly = FALSE)

## S4 method for signature 'ModDNAStringSet'
alphabetFrequency(x, as.prob = FALSE, collapse = FALSE, baseOnly = FALSE)

## S4 method for signature 'ModRNAStringSet'
alphabetFrequency(x, as.prob = FALSE, collapse = FALSE, baseOnly = FALSE)

## S4 method for signature 'MaskedModString'
alphabetFrequency(x, as.prob = FALSE, ...)

## S4 method for signature 'ModStringViews'
letterFrequency(x, letters, OR = "|", as.prob = FALSE, ...)

## S4 method for signature 'MaskedModString'
letterFrequency(x, letters, OR = "|", as.prob = FALSE)

## S4 method for signature 'ModStringSet'
```

```
consensusMatrix(x, as.prob = FALSE, shift = 0L, width = NULL, baseOnly = FALSE)

## S4 method for signature 'ModDNAStringSet'
consensusString(x, threshold = 0.25, shift = 0L, width = NULL)

## S4 method for signature 'ModRNAStringSet'
consensusString(x, threshold = 0.25, shift = 0L, width = NULL)

## S4 method for signature 'ModStringViews'
consensusString(x, threshold, shift = 0L, width = NULL)
```

Arguments

x	a [ModString](), a [ModStringSet](), a [ModStringViews]() or a [MaskedModString]() object.
as.prob	TRUE or FALSE (default): Should the result be returned as probabilities instead of counts? (sum per column = 1)
baseOnly	TRUE or FALSE (default): Should the result omit occurances of the letters N.-+?
collapse	TRUE or FALSE (default): Should the results summed up all elements for ModStringSet or ModStringViews objects or reported per element.
...	See [letterFrequency]().
letters	See [letterFrequency]().
OR	See [letterFrequency]().
shift	See [letterFrequency]().
width	See [letterFrequency]().
threshold	Since the amiguityMap is fixed to "?" for ModString objects, only the treshold can be set (default threshold = 0.25)

Value

a matrix with the results (letter x pos).

Examples

```
mod <- ModDNAString(paste(alphabet(ModDNAString()), collapse = ""))
mod
hasOnlyBaseLetters(mod)
alphabetFrequency(mod)
```

MaskedModString   *MaskedModString objects*

Description

The functions are implemented as defined in the Biostrings package. Have a look the [MaskedXString]() class.

Usage

```
## S4 method for signature 'MaskedModString'
seqtype(x)
```

Arguments

x   a ModString object.

Value

a MaskedModString object.

Examples

```
# Mask positions
mask <- Mask(mask.width=5, start=c(2), width=c(3))
mr <- ModRNAString("ACGU7")
mr

masks(mr) <- mask
mr

# Invert masks
mr <- gaps(mr)
mr

# Drop the mask
masks(mr) <- NULL
mr
```

ModDNAString   *ModDNAString class*

Description

A ModDNAString object allows DNA sequences with modified nucleotides to be stored and manipulated.

Usage

```
ModDNAString(x = "", start = 1, nchar = NA)
```

Arguments

x   the input as a character.

start   the postion in the character vector to use as start position in the ModDNAString
                object (default start = 1).

nchar   the width of the character vector to use in the ModDNAString object (default
                nchar = NA). The end position is calculated as start + nchar - 1.

Details

The ModDNAString class contains the virtual `ModString` class, which is itself based on the `XString` class. Therefore, functions for working with XString classes are inherited.

The `alphabet` of the ModDNAString class consist of the non-extended IUPAC codes "A,G,C,T,N", the gap letter "-", the hard masking letter "+", the not available letter "." and letters for individual modifications: alphabet(ModDNAString()).

Since the special characters are encoded differently depending on the OS and encoding settings of the R session, it is not always possible to enter a DNA sequence containing modified nucleotides via the R console. The most convinient solution for this problem is to use the function `modifyNucleotides` and modify and existing DNAString or ModDNAString object.

A ModDNAString object can be converted into a `DNAString` object using the `DNAstring()` constructor. Modified nucleotides are automaitcally converted intro their base nucleotides.

If a modified DNA nucleotide you want to work with is not part of the alphabet, please let us know.

Value

a `ModDNAString` object

Examples

```
# Constructing ModDNAString containing an m6A
md1 <- ModDNAString("AGCT`")
md1

# the alphabet of the ModDNAString class
alphabet(md1)
# due to encoding issues the shortNames can also be used
shortName(md1)
# due to encoding issues the nomenclature can also be used
nomenclature(md1)

# convert to DNAString
d1 <- DNAString(md1)
d1
```

---

modifyNucleotides   *Modifying nucleotides in a nucleotide sequence (or set of sequences)*
                                at specified locations

Description

`modifyNucleotides` modifies a nucleotide in a sequence (or set of sequences) based on the type of modification provided. It checks for the identity of the base nucleotide to be

Usage

```
modifyNucleotides(
  x,
  at,
  mod,
  nc.type = "short",
```

```
      stop.on.error = TRUE,
      verbose = FALSE
    )

    ## S4 method for signature 'ModString'
    modifyNucleotides(
      x,
      at,
      mod,
      nc.type = c("short", "nc"),
      stop.on.error = TRUE,
      verbose = FALSE
    )

    ## S4 method for signature 'ModStringSet'
    modifyNucleotides(
      x,
      at,
      mod,
      nc.type = c("short", "nc"),
      stop.on.error = TRUE,
      verbose = FALSE
    )

    ## S4 method for signature 'DNAString'
    modifyNucleotides(
      x,
      at,
      mod,
      nc.type = c("short", "nc"),
      stop.on.error = TRUE,
      verbose = FALSE
    )

    ## S4 method for signature 'RNAString'
    modifyNucleotides(
      x,
      at,
      mod,
      nc.type = c("short", "nc"),
      stop.on.error = TRUE,
      verbose = FALSE
    )

    ## S4 method for signature 'DNAStringSet'
    modifyNucleotides(
      x,
      at,
      mod,
      nc.type = c("short", "nc"),
      stop.on.error = TRUE,
      verbose = FALSE
```

```
)

## S4 method for signature 'RNAStringSet'
modifyNucleotides(
  x,
  at,
  mod,
  nc.type = c("short", "nc"),
  stop.on.error = TRUE,
  verbose = FALSE
)
```

Arguments

x	a [ModString](#) or [ModStringSet](#) object
at	the location where the modification should be made.
	The same input as in the original [replaceLetterAt](#) are expected:
	If x is a [ModString](#) object, then at is typically an integer vector with no NAs but a logical vector or Rle object is valid too. Locations can be repeated and in this case the last replacement to occur at a given location prevails.
	If x is a rectangular [ModStringSet](#) object, then at must be a matrix of logicals with the same dimensions as x. If the [ModStringSet](#) is not rectangular, at must be a list of logical vectors.
mod	The modification short name or nomenclature
	If x is a [ModString](#) object, then letter must be a [ModString](#) object or a character vector (with no NA) with a total number of letters (sum(nchar(letter))) equal to the number of locations specified in at.
	If x is a rectangular [ModStringSet](#) object, then letter must be a [ModStringSet](#) object, a list of character vectors or a [CharacterList](#) of the same length as x. In addition, the number of letters in each element of letter must match the number of locations specified in the corresponding row of at (all(width(letter) == rowSums(at))).
nc.type	the type of nomenclature to be used. Either "short" or "nc". "Short" for m3C would be "m3C", "nc" for m3C would be "3C". (default = "short")
stop.on.error	For combineIntoModstrings: TRUE(default) or FALSE: Should an error be raised upon encounter of incompatible positions?
verbose	See [replaceLetterAt](#).

Value

the input [ModString](#) or [ModStringSet](#) object with the changes applied

Examples

```
# modify nucleotides in a ModDNAString
seq <- ModDNAString("AGTC")
seq

mseq1 <- modifyNucleotides(seq,c(1,2,4),c("1mA","7mG","3mC"))
mseq1

# This fails since m7G requires a G at the selected position in the sequence
```

```
## Not run:
mseq <- modifyNucleotides(seq,c(3),c("7mG"))

## End(Not run)


# modify nucleotides in a ModRNAString
seq <- ModRNAString("AGUC")
seq

mseq1 <- modifyNucleotides(seq,c(1,2,4),c("m1A","m7G","m3C"))
mseq1

# This fails since m7G requires a G at the selected position in the sequence
## Not run:
mseq <- modifyNucleotides(seq,c(3),c("m7G"))

## End(Not run)
```

ModRNAString   *ModDNAString class*

Description

A `ModRNAString` object allows RNA sequences with modified nucleotides to be stored and manipulated.

Usage

```
ModRNAString(x = "", start = 1, nchar = NA)
```

Arguments

x	the input as a `character`.
start	the postion in the character vector to use as start position in the `ModRNAString` object (default `start = 1`).
nchar	the width of the character vector to use in the `ModRNAString` object (default `nchar = NA`). The end position is calculated as `start + nchar - 1`.

Details

The ModRNAString class contains the virtual [ModString](#) class, which is itself based on the [XString](#) class. Therefore, functions for working with XString classes are inherited.

The alphabet of the ModRNAString class consist of the non-extended IUPAC codes "A,G,C,U", the gap letter "-", the hard masking letter "+", the not available letter "." and letters for individual modifications: `alphabet(ModRNAString())`.

Since the special characters are encoded differently depending on the OS and encoding settings of the R session, it is not always possible to enter a RNA sequence containing modified nucleotides via the R console. The most convinient solution for this problem is to use the function [modifyNucleotides](#) and modify and existing RNAString or ModRNAString object.

A `ModRNAString` object can be converted into a `RNAString` object using the `RNAstring()` constructor. Modified nucleotides are automaitcally converted intro their base nucleotides.

If a modified RNA nucleotide you want to work with is not part of the alphabet, please let us know.

Value

a `ModRNAString` object

Examples

```
# Constructing ModDNAString containing an m6A and a dihydrouridine
mr1 <- ModRNAString("AGCU`D")
mr1

# the alphabet of the ModRNAString class
alphabet(mr1)
# due to encoding issues the shortNames can also be used
shortName(mr1)
# due to encoding issues the nomenclature can also be used
nomenclature(mr1)

# convert to RNAString
r1 <- RNAString(mr1)
r1
```

ModString   *ModString objects*

Description

The virtual `ModString` class derives from the `XString` virtual class. Like its parent and its children,
it is used for storing sequences of characters. However, the `XString`/`BString` class requires single
byte characters as the letters of the input sequences. The `ModString` extends the capability for
multi-byte chracters by encoding these characters into a single byte characters using a dictionary
for internal conversion. It also takes care of different encoding behavior of operating systems.

The [ModDNAString](#) and [ModRNAString](#) classes derive from the `ModString` class and use the func-
tionality to store nucleotide sequences containing modified nucleotides. To describe modified RNA
and DNA nucleotides with a single letter, special characters are commonly used, eg. from the greek
alphabet, which are multi-byte characters.

The `ModString` class is virtual and it cannot be directly used to create an object. Please have a look
at [ModDNAString](#) and [ModRNAString](#) for the specific alphabets of the individual classes.

Modstrings   *Modstrings: implementation of Biostrings to work with nucleotide se-*
                                 quences containing modified nucleotides.

Description

Representing nucleotide modifications in a nucleotide sequence is usually done via special charac-
ters from a number of sources. This represents a challenge to work with in R and the `Biostrings`
package. The `Modstrings` package implements this functionallity for RNA and DNA sequences
containing modified nucleotides by translating the character internally in order to work with the
infrastructure of the `Biostrings` package. For this the `ModRNAString` and `ModDNAString` classes
and derivates and functions to construct and modify these objects despite the encoding issues are

implemenented. In addition the conversion from sequences to list like location information (and the reverse operation) is implemented as well.

A good place to start would be the vignette and the man page for the ModStringSet objects.

The alphabets for the modifications used in this package are based on the compilation of RNA modifications by http://modomics.genesilico.pl by the Bujnicki lab and DNA modifications https://dnamod.hoffmanlab.org by the Hoffman lab. Both alphabets were modified to remove some incompatible characters.

Author(s)

Felix G M Ernst [aut,cre] and Denis L.J. Lafontaine [ctb]

Modstrings-internals   *Modstrings internals*

Description

Analog to Biostrings there are a few functions, which should only be used internally. Otherwise take care.

Usage

```
## S4 method for signature 'ModDNAString'
seqtype(x)

## S4 method for signature 'ModRNAString'
seqtype(x)

## S4 replacement method for signature 'ModString'
seqtype(x) <- value

## S4 method for signature 'ModString'
XString(seqtype, x, start = NA, end = NA, width = NA)

## S4 replacement method for signature 'ModStringSet'
seqtype(x) <- value

## S4 method for signature 'ModStringSet'
XStringSet(seqtype, x, start = NA, end = NA, width = NA, use.names = TRUE)

data(modsRNA)

data(modsDNA)

data(MOD_RNA_DICT_MODOMICS)

data(MOD_RNA_DICT_TRNADB)
```

Arguments

seqtype, x, start, end, width, use.names, value
              used internally

Format

An object of class DFrame with 162 rows and 9 columns.

An object of class DFrame with 47 rows and 5 columns.

An object of class DFrame with 170 rows and 3 columns.

An object of class DFrame with 60 rows and 3 columns.

Value

a XString* object

ModStringSet   *ModStringSet objects*

Description

The ModStringSet class is a container for storing a set of ModString objects. It follows the same principles as the other XStringSet objects.

As usual the ModStringSet containers derive directly from the XStringSet virtual class.

The ModStringSet class is in itself a virtual class with two types of derivates:

- ModDNAStringSet
- ModRNAStringSet

Each class can only be converted to its parent DNAStringSet or RNAStringSet. The modified nucleotides will be converted to their original nucleotides.

Please note, that due to encoding issues not all modifications can be instanciated directly from the console. The vignette contains a comphrensive explanation and examples for working around the problem.

Usage

```
ModDNAStringSet(
  x = character(),
  start = NA,
  end = NA,
  width = NA,
  use.names = TRUE
)

ModRNAStringSet(
  x = character(),
  start = NA,
  end = NA,
  width = NA,
  use.names = TRUE
)
```

Arguments

x	Either a character vector (with no NAs), or an ModString, ModStringSet or ModStringViews object.
start, end, width	
	Either NA, a single integer, or an integer vector of the same length as x specifying how x should be "narrowed" (see ?narrow for the details).
use.names	TRUE or FALSE. Should names be preserved?

Value

a `ModStringSet` object.

Examples

```
# Constructing ModDNAStringSet containing an m6A
m1 <- ModDNAStringSet(c("AGCT`","AGCT`"))
m1

# converting to DNAStringSet

# Constructing ModRNAStringSet containing an m6A
m2 <- ModRNAStringSet(c("AGCU`","AGCU`"))
m2
```

ModStringSet-io   *Read/write an ModStringSet object from/to a file*

Description

Functions to read/write an ModStringSet object from/to a file.

Usage

```
readModDNAStringSet(
  filepath,
  format = "fasta",
  nrec = -1L,
  skip = 0L,
  seek.first.rec = FALSE,
  use.names = TRUE,
  with.qualities = FALSE
)

readModRNAStringSet(
  filepath,
  format = "fasta",
  nrec = -1L,
  skip = 0L,
  seek.first.rec = FALSE,
  use.names = TRUE,
  with.qualities = FALSE
```

```
)

writeModStringSet(
  x,
  filepath,
  append = FALSE,
  compress = FALSE,
  compression_level = NA,
  format = "fasta",
  ...
)
```

Arguments

filepath, format, nrec, skip, seek.first.rec, use.names, with.qualities,
append, compress, compression_level, ...
                See [XStringSet-io](#) for more details.

x   A ModStringSet object.

Value

A [ModStringSet](#) of the defined type.

Examples

```
seqs <- paste0(paste(alphabet(ModDNAString()), collapse = ""),
               c("A","G","T"))
seqs

set <- ModDNAStringSet(seqs)
set

file <- tempfile()
writeModStringSet(set, file)
read <- readModDNAStringSet(file)
read
```

ModStringSetList   *ModStringSetList*

Description

title

Usage

```
ModDNAStringSetList(..., use.names = TRUE)

ModRNAStringSetList(..., use.names = TRUE)
```

Arguments

`...`	[ModStringSet](#) objects of one type.
`use.names`	TRUE(default) or FALSE: Whether names of the input `ModStringSet` objects should be stored and used as the element names in the `ModStringSetList`.

Value

a `ModStringSetList` object.

Examples

```
mrseq <- c("ACGU7","ACGU7","ACGU7","ACGU7")
mrseq

# Example: contruction of ModStringSetlist from ModString objects
mr <- ModRNAString("ACGU7")
mr

mrs <- ModRNAStringSet(list(mr,mr,mr,mr))
mrs

mrsl <- ModRNAStringSetList(mrs,mrs)
mrsl

# Example: construction of ModStringSetlist from mixed sources
mrsl2 <- ModRNAStringSetList(mrs,mrseq)
mrsl2
```

ModStringViews *The ModStringViews class extending the XStringViews class*

Description

As the [XStringViews](#) the ModStringViews is the basic container for storing a set of views on the same sequence (this time a ModString object).

Usage

```
## S4 method for signature 'ModString'
Views(subject, start = NULL, end = NULL, width = NULL, names = NULL)
```

Arguments

subject, start, end, width, names
    See [XStringViews](#).

Details

For the details have a look at the [XStringViews](#) class.

Value

a `ModStringViews` object.

Examples

```
seq <- ModDNAString("AGC6AGC6")
seq

v <- Views(seq, start = 3:1, end = 6:8)
v
```

QualityScaledModStringSet

QualityScaledModDNAStringSet   and   QualityScaledModRNAS-
tringSet objects

Description

title

Usage

```
QualityScaledModDNAStringSet(x, quality)

QualityScaledModRNAStringSet(x, quality)

readQualityScaledModDNAStringSet(
  filepath,
  quality.scoring = c("phred", "solexa", "illumina"),
  nrec = -1L,
  skip = 0L,
  seek.first.rec = FALSE,
  use.names = TRUE
)

readQualityScaledModRNAStringSet(
  filepath,
  quality.scoring = c("phred", "solexa", "illumina"),
  nrec = -1L,
  skip = 0L,
  seek.first.rec = FALSE,
  use.names = TRUE
)

writeQualityScaledModStringSet(
  x,
  filepath,
  append = FALSE,
  compress = FALSE,
  compression_level = NA
)
```

Arguments

x
: For the QualityScaled*StringSet constructors: Either a character vector, or an ModString, ModStringSet or ModStringViews object.

  For writeQualityScaledXStringSet: A QualityScaledModDNAStringSet or QualityScaledModRNAStringSet object.

quality
: A XStringQuality object.

filepath, nrec, skip, seek.first.rec, use.names, append, compress, compression_level
: See QualityScaledXStringSet-class.

quality.scoring
: Specify the quality scoring used in the FASTQ file. Must be one of "phred" (the default), "solexa", or "illumina". If set to " phred" (or "solexa" or "illumina"), the qualities will be stored in a PhredQuality (or SolexaQuality or IlluminaQuality, respectively) object.

Value

a QualityScaledModDNAStringSet or QualityScaledModDNAStringSet object

Examples

```
seq <- ModRNAString("AGCU7")
seq

qseq <- PhredQuality(paste0(rep("!", length(seq)), collapse = ""))
qseq

qset <- QualityScaledModRNAStringSet(seq, qseq)
qset
```

---

replaceLetterAt	*Replacing letters in a nucleotide sequence (or set of nucleotide sequences) at some specified locations containing nucleotide modifications*

Description

replaceLetterAt replaces a letter in a ModString objects with a new letter. In contrast to modifyNucleotides it does not check the letter to be replaced for its identity, it just replaces it and behaves exactly like the

Usage

```
## S4 method for signature 'ModString'
replaceLetterAt(x, at, letter, verbose = FALSE)

## S4 method for signature 'ModStringSet'
replaceLetterAt(x, at, letter, verbose = FALSE)
```

Arguments

x	a ModString or ModStringSet object
at	the location where the replacement should be made.
	The same input as in replaceLetterAt are expected:
	If x is a ModString object, then at is typically an integer vector with no NAs but a logical vector or Rle object is valid too. Locations can be repeated and in this case the last replacement to occur at a given location prevails.
	If x is a rectangular ModStringSet object, then at must be a matrix of logicals with the same dimensions as x. If the ModStringSet is not rectangular, at must be a list of logical vectors.
letter	The new letters.
	The same input as in replaceLetterAt are expected:
	If x is a ModString object, then letter must be a ModString object or a character vector (with no NAs) with a total number of letters (sum(nchar(letter))) equal to the number of locations specified in at.
	If x is a rectangular ModStringSet object, then letter must be a ModStringSet object or a character vector of the same length as x. In addition, the number of letters in each element of letter must match the number of locations specified in the corresponding row of at (all(width(letter) == rowSums(at))).
verbose	See replaceLetterAt.

Value

the input ModString or ModStringSet object with the changes applied

Examples

```
# Replacing the last two letters in a ModDNAString
seq1 <- ModDNAString("AGTC")
seq
seq2 <- replaceLetterAt(seq1,c(3,4),"CT")
seq2

# Now containg and m3C
seq2 <- replaceLetterAt(seq1,c(3,4),ModDNAString("/T"))
seq2

# Replacing the last two letters in a set of sequences
set1 <- ModDNAStringSet(c("AGTC","AGTC"))
set1

set2 <- replaceLetterAt(set1,
                        matrix(rep(c(FALSE,FALSE,TRUE,TRUE),2),
                               nrow = 2,
                               byrow = TRUE),
                        c("CT","CT"))
set2
```

---

sanitizeInput   *Sanitize input strings for use with ModString classes*

Description

Since the one letter nomenclature for RNA and DNA modification differs depending on the source, a translation to a common alphabet is necessary.

sanitizeInput exchanges based on a dictionary. The dictionary is expected to be a DataFrame with two columns, mods_abbrev and short_name. Based on the short_name the characters from in the input are converted from values of mods_abbrev into the the ones from alphabet.

Only different values will be searched for and exchanged.

sanitizeFromModomics and sanitizeFromtRNAdb use a predefined dictionary, which is builtin.

Usage

```
sanitizeInput(input, dictionary)

sanitizeFromModomics(input)

sanitizeFromtRNAdb(input)
```

Arguments

input	a character vector, which should be converted
dictionary	a DataFrame containing at least two columns mods_abbrev and short_name. From this a dictionary table is contructed for exchaning old to new letters.

Value

the modified character vector compatible for constructing a ModString object.

Examples

```
# Modomics
chr <- "AGC@"
# Error since the @ is not in the alphabet
## Not run:
seq <- ModRNAString(chr)

## End(Not run)
seq <- ModRNAString(sanitizeFromModomics(chr))
seq

# tRNAdb
chr <- "AGC+"
# No error but the + has a different meaning in the alphabet
## Not run:
seq <- ModRNAString(chr)

## End(Not run)
seq <- ModRNAString(sanitizeFromtRNAdb(chr))
seq
```

---

separate *Separating and combining a modification information into/from a* XString *and a* GRanges *object*

Description

With combineIntoModstrings and separate the construction and deconstruction of ModString Objects from an interacive session avoiding problematic encoding issues. In addition, modification information can be transfered from/to tabular data with these functions.

combineIntoModstrings expects seqnames(gr) or names(gr) to match the available names(x). Only information with strand information * and + are used.

separate when used with a GRanges/GRangesList object will return an object of the same type, but with modifications seperated. For example an element with mod = "m1Am" will be returned as two elements with mod = c("Am","m1A"). The reverse operation is available via combineModifications().

removeIncompatibleModifications filters incompatible modification from a GRanges or GRangesList. incompatibleModifications() returns the logical vector used for this operation.

Usage

```
separate(x, nc.type = "short")

combineIntoModstrings(
  x,
  gr,
  with.qualities = FALSE,
  quality.type = "Phred",
  stop.on.error = TRUE,
  verbose = FALSE,
  ...
)

combineModifications(gr, ...)

incompatibleModifications(gr, x, ...)

removeIncompatibleModifications(gr, x, ...)

## S4 method for signature 'ModString'
separate(x, nc.type = c("short", "nc"))

## S4 method for signature 'ModStringSet'
separate(x, nc.type = c("short", "nc"))

## S4 method for signature 'GRanges'
separate(x)

## S4 method for signature 'GRangesList'
separate(x)

## S4 method for signature 'XString,GRanges'
```

```
combineIntoModstrings(
  x,
  gr,
  with.qualities = FALSE,
  quality.type = "Phred",
  stop.on.error = TRUE,
  verbose = FALSE,
  ...
)

## S4 method for signature 'XStringSet,GRangesList'
combineIntoModstrings(
  x,
  gr,
  with.qualities = FALSE,
  quality.type = "Phred",
  stop.on.error = TRUE,
  verbose = FALSE,
  ...
)

## S4 method for signature 'XStringSet,GRanges'
combineIntoModstrings(
  x,
  gr,
  with.qualities = FALSE,
  quality.type = "Phred",
  stop.on.error = TRUE,
  verbose = FALSE,
  ...
)

## S4 method for signature 'GRanges'
combineModifications(gr)

## S4 method for signature 'GRangesList'
combineModifications(gr)

## S4 method for signature 'GRanges,XString'
incompatibleModifications(gr, x)

## S4 method for signature 'GRanges,XStringSet'
incompatibleModifications(gr, x)

## S4 method for signature 'GRangesList,XStringSet'
incompatibleModifications(gr, x)

## S4 method for signature 'GRanges,XString'
removeIncompatibleModifications(gr, x)

## S4 method for signature 'GRanges,XStringSet'
removeIncompatibleModifications(gr, x)
```

```
## S4 method for signature 'GRangesList,XStringSet'
removeIncompatibleModifications(gr, x)
```

Arguments

x	For separate: a ModString/ModStringSet or GRanges/GRangesListobject
	For combineIntoModstrings: a XString and a XStringSet object.
nc.type	the type of nomenclature to be used. Either "short" or "nc". "Short" for m3C would be "m3C", "nc" for m3C would be "3C". (default = "short")
gr	a GRanges object
with.qualities	TRUE or FALSE (default): Should the values from a score column of the GRanges object stored? If set with.qualities = TRUE, combineIntoModstrings will try to construct a QualityScaledModStringSet object.
quality.type	the type of QualityXStringSet used, if with.qualities = TRUE. Must be on of the following values: "Phred","Solexa","Illumina".
stop.on.error	For combineIntoModstrings: TRUE(default) or FALSE: Should an error be raised upon encounter of incompatible positions?
verbose	For combineIntoModstrings: TRUE or FALSE (default): Should verbose information reported on the positions filled with modifications? This settings is passed onto modifyNucleotides.
...	default.quality: for combineIntoModstrings: the default.quality default value for non-modified positions. (default: default.quality = 0L)

Value

for separate a GRanges object and for combineIntoModstrings a ModString* object or a QualityScaledModStringSe if with.qualities = TRUE.

Examples

```
library(GenomicRanges)
# ModDNAString
seq <- ModDNAString(paste(alphabet(ModDNAString()), collapse = ""))
seq

gr <- separate(seq)
gr

seq2 <- combineIntoModstrings(as(seq,"DNAString"),gr)
seq2

seq == seq2
# ModRNAString
seq <- ModRNAString(paste(alphabet(ModRNAString()), collapse = ""))
seq

gr <- separate(seq)
gr

# Separating RNA modifications
gr <- gr[1]
separate(gr)
```

```
# ... and combine them again (both operations work only on a subset of
# modifications)
combineModifications(separate(gr))

# handling incompatible modifications
seq <- RNAString("AGCU")
gr <- GRanges(c("chr1:1:+","chr1:2:+"),mod="m1A")
incompatibleModifications(gr,seq)

#
removeIncompatibleModifications(gr,seq)
```

---

shortName | *Base information for sequence characters of nucleotide strings containing modifications*

Description

The `alphabet()`, `shortName()` `fullName()` and `nomenclature()` functions return the letters, names and associated abbreviations for the type of ModString. `alphabet()` returns the normal letters and modification letters, whereas `shortName()`, `fullName()` and `nomenclature()` return results for modifications only.

Usage

```
shortName(x)

fullName(x)

nomenclature(x)

## S4 method for signature 'ModString'
alphabet(x, baseOnly = FALSE)

## S4 method for signature 'ModStringSet'
alphabet(x, baseOnly = FALSE)

## S4 method for signature 'ModString'
shortName(x)

## S4 method for signature 'ModStringSet'
shortName(x)

## S4 method for signature 'ModString'
fullName(x)

## S4 method for signature 'ModStringSet'
fullName(x)

## S4 method for signature 'ModString'
nomenclature(x)
```

```
## S4 method for signature 'ModStringSet'
nomenclature(x)
```

Arguments

x	a `ModString` or `ModStringSet` object
baseOnly	`TRUE` or `FALSE` (default): Should the result omit occurances of the letters `N.-+`?

Value

a character vector.

Examples

```
alphabet(ModDNAString())
shortName(ModDNAString())
nomenclature(ModDNAString())
```

Index