

# Package ‘Modstrings’

February 27, 2021

**Type** Package

**Title** Working with modified nucleotide sequences

**Version** 1.7.1

**Date** 2020-12-13

**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 functionality 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 derivatives and functions to construct and modify these objects despite the encoding issues are implemented. 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** 7.1.1

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

**git\_url** <https://git.bioconductor.org/packages/Modstrings>

**git\_branch** master

**git\_last\_commit** b8093fe

**git\_last\_commit\_date** 2020-12-13

**Date/Publication** 2021-02-26

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

**Maintainer** Felix G.M. Ernst <[felix.gm.ernst@outlook.com](mailto:felix.gm.ernst@outlook.com)>

## R topics documented:

letterFrequency . . . . .	2
MaskedModString . . . . .	4
ModDNAStrng . . . . .	5
modifyNucleotides . . . . .	6
ModRNAStrng . . . . .	8
ModString . . . . .	9
Modstrings . . . . .	10
Modstrings-internals . . . . .	10
ModStringSet . . . . .	11
ModStringSet-io . . . . .	12
ModStringSetList . . . . .	14
ModStringViews . . . . .	14
QualityScaledModStringSet . . . . .	15
replaceLetterAt . . . . .	16
sanitizeInput . . . . .	17
separate . . . . .	18
shortName . . . . .	21
<b>Index</b>	<b>23</b>

---

letterFrequency	<i>Calculate the frequency of letters in nucleotide sequence with modifications, or the consensus matrix of a set of sequences</i>
-----------------	--

---

## Description

These functions follow the same principle as the [Biostrings](#) functions. Please be aware, that the matrices 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 <a href="#">ModString</a> , a <a href="#">ModStringSet</a> , a <a href="#">ModStringViews</a> or a <a href="#">MaskedModString</a> 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 occurrences of the letters N. -+?
collapse	TRUE or FALSE (default): Should the results summed up all elements for <a href="#">ModStringSet</a> or <a href="#">ModStringViews</a> objects or reported per element.
...	See <a href="#">letterFrequency</a> .
letters	See <a href="#">letterFrequency</a> .

OR	See <a href="#">letterFrequency</a> .
shift	See <a href="#">letterFrequency</a> .
width	See <a href="#">letterFrequency</a> .
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 = ""))
hasOnlyBaseLetters(mod)
alphabetFrequency(mod)
```

---

MaskedModString	<i>MaskedModString objects</i>
-----------------	--------------------------------

---

**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")
masks(mr) <- mask

# Invert masks
mr <- gaps(mr)

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

---

ModDNAStrng	<i>ModDNAStrng class</i>
-------------	--------------------------

---

### Description

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

### Usage

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

### Arguments

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

### Details

The ModDNAStrng 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 ModDNAStrng 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(ModDNAStrng())`.

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 convenient solution for this problem is to use the function [modifyNucleotides](#) and modify an existing DNAStrng or ModDNAStrng object.

A ModDNAStrng object can be converted into a DNAStrng object using the `DNAStrng()` constructor. Modified nucleotides are automatically converted into 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 ModDNAStrng object

### Examples

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

# the alphabet of the ModDNAStrng 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)
```

---

modifyNucleotides	<i>Modifying nucleotides in a nucleotide sequence (or set of sequences) at specified locations</i>
-------------------	--

---

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

### Value

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

### Examples

```
# modify nucleotides in a ModDNAString
seq <- ModDNAString("AGTC")
mseq1 <- modifyNucleotides(seq,c(1,2,4),c("1mA","7mG","3mC"))
# 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")
mseq1 <- modifyNucleotides(seq,c(1,2,4),c("1mA","7mG","3mC"))
# 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)
```

---

ModRNAString	<i>ModDNAString class</i>
--------------	---------------------------

---

### Description

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

### Usage

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

### Arguments

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



## 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 into 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")

# 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)
```

---

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 characters 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 functionality 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 sequences containing modified nucleotides.*


---

## 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 functionality 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 derivatives and functions to construct and modify these objects despite the encoding issues are implemented. 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 `DataFrame` with 162 rows and 9 columns.

An object of class `DataFrame` with 47 rows and 5 columns.

An object of class `DataFrame` with 170 rows and 3 columns.

An object of class `DataFrame` with 60 rows and 3 columns.

### Value

a `XString*` object

---

ModStringSet	<i>ModStringSet objects</i>
--------------	-----------------------------

---

### 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 derivatives:

- `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 instantiated directly from the console. The vignette contains a comprehensive 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"))  
set <- ModDNAStringSet(seqs)  
file <- tempfile()  
writeModStringSet(set, file)  
read <- readModDNAStringSet(file)
```

---

ModStringSetList	<i>ModStringSetList</i>
------------------	-------------------------

---

**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")

# Example: construction of ModStringSetlist from ModString objects
mr <- ModRNAString("ACGU7")
mrs <- ModRNAStringSet(list(mr,mr,mr,mr))
mrsl <- ModRNAStringSetList(mrs,mrs)

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

---

ModStringViews	<i>The ModStringViews class extending the XStringViews class</i>
----------------	--

---

**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 <- ModDNString("AGC6AGC6")
v <- Views(seq, start = 3:1, end = 6:8)
```

---

QualityScaledModStringSet

*QualityScaledModDNStringSet* and *QualityScaledModRNAS-*  
*tringSet* objects

---

**Description**

title

**Usage**

```
QualityScaledModDNStringSet(x, quality)
```

```
QualityScaledModRNStringSet(x, quality)
```

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

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

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

```

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 `QualityScaledModRNAStringSet` object

---

replaceLetterAt	<i>Replacing letters in a nucleotide sequence (or set of nucleotide sequences) at some specified locations containing nucleotide modifications</i>
-----------------	--

---

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

**Value**

the input `ModString` or `ModStringSet` object with the changes applied

**Examples**

```
# Replacing the last two letters in a ModDNAString
seq1 <- ModDNAString("AGTC")
seq2 <- replaceLetterAt(seq1,c(3,4),"CT")
# Now containg and m3C
seq2 <- replaceLetterAt(seq1,c(3,4),ModDNAString("/T"))
# Replacing the last two letters in a set of sequences
set1 <- ModDNAStringSet(c("AGTC","AGTC"))
set2 <- replaceLetterAt(set1,
                        matrix(rep(c(FALSE,FALSE,TRUE,TRUE),2),
                              nrow = 2,
                              byrow = TRUE),
                        c("CT","CT"))
```

---

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 `sanitizeFrommRNadb` 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 constructed for exchanging 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))

# 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))

```

---

separate	<i>Separating and combining a modification information into/from a XString and a GRanges object</i>
----------	---

---

**Description**

With `combineIntoModstrings` and `separate` the construction and deconstruction of `ModString` Objects from an interactive session avoiding problematic encoding issues. In addition, modification information can be transferred 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 separated. 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/GRangesList object 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 <a href="#">QualityScaledModStringSet</a> object.
quality.type	the type of QualityXStringSet used, if with.qualities = TRUE. Must be one 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 QualityScaledModStrings 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 occurrences of the letters N. -+?

**Value**

a character vector.

**Examples**

```
alphabet(ModDNAStrng())
shortName(ModDNAStrng())
nomenclature(ModDNAStrng())
```

# Index

## \* datasets

- Modstrings-internals, [10](#)
- `==`, ModString, ModString-method (ModString), [9](#)
- `==`, ModString, XString-method (ModString), [9](#)
- `==`, ModStringSet, ModStringSet-method (ModStringSet), [11](#)
- `==`, ModStringSet, XStringSet-method (ModStringSet), [11](#)
- `==`, ModStringViews, ModStringViews-method (ModStringViews), [14](#)
- `==`, ModStringViews, XString-method (ModStringViews), [14](#)
- `==`, XString, ModString-method (ModString), [9](#)
- `==`, XStringSet, ModStringSet-method (ModStringSet), [11](#)
- `==`, XStringViews, ModString-method (ModStringViews), [14](#)
  
- alphabet, [5](#)
- alphabet (shortName), [21](#)
- alphabet, ModString-method (shortName), [21](#)
- alphabet, ModStringSet-method (shortName), [21](#)
- alphabetFrequency (letterFrequency), [2](#)
- alphabetFrequency, MaskedModString-method (letterFrequency), [2](#)
- alphabetFrequency, ModDNAString-method (letterFrequency), [2](#)
- alphabetFrequency, ModDNAStringSet-method (letterFrequency), [2](#)
- alphabetFrequency, ModRNAString-method (letterFrequency), [2](#)
- alphabetFrequency, ModRNAStringSet-method (letterFrequency), [2](#)
- as.character, ModString-method (ModString), [9](#)
- as.character, ModStringSet-method (ModStringSet), [11](#)
- as.vector, ModString-method (ModString), [9](#)
  
- Biostrings, [2](#)
  
- CharacterList, [7](#)
- combineIntoModstrings (separate), [18](#)
- combineIntoModstrings, XString, GRanges-method (separate), [18](#)
- combineIntoModstrings, XStringSet, GRanges-method (separate), [18](#)
- combineIntoModstrings, XStringSet, GRangesList-method (separate), [18](#)
- combineModifications (separate), [18](#)
- combineModifications, GRanges-method (separate), [18](#)
- combineModifications, GRangesList-method (separate), [18](#)
- consensusMatrix, ModStringSet-method (letterFrequency), [2](#)
- consensusString (letterFrequency), [2](#)
- consensusString, ModDNAStringSet-method (letterFrequency), [2](#)
- consensusString, ModRNAStringSet-method (letterFrequency), [2](#)
- consensusString, ModStringViews-method (letterFrequency), [2](#)
  
- fullName (shortName), [21](#)
- fullName, ModString-method (shortName), [21](#)
- fullName, ModStringSet-method (shortName), [21](#)
  
- hasOnlyBaseLetters (letterFrequency), [2](#)
- hasOnlyBaseLetters, ModDNAString-method (letterFrequency), [2](#)
- hasOnlyBaseLetters, ModDNAStringSet-method (letterFrequency), [2](#)
- hasOnlyBaseLetters, ModRNAString-method (letterFrequency), [2](#)
- hasOnlyBaseLetters, ModRNAStringSet-method (letterFrequency), [2](#)
  
- incompatibleModifications (separate), [18](#)
- incompatibleModifications, GRanges, XString-method (separate), [18](#)

- incompatibleModifications, GRanges, XStringSet-method (separate), 18
- incompatibleModifications, GRangesList, XStringSet-method (separate), 18
- letterFrequency, 2, 3, 4
- letterFrequency, MaskedModString-method (letterFrequency), 2
- letterFrequency, ModStringViews-method (letterFrequency), 2
- letterFrequencyInSlidingView (letterFrequency), 2
- MaskedModString, 3, 4
- MaskedXString, 4
- MOD\_RNA\_DICT\_MODALOMICS (Modstrings-internals), 10
- MOD\_RNA\_DICT\_TRNADB (Modstrings-internals), 10
- ModDNAString, 5, 9
- ModDNAString-class (ModString), 9
- ModDNAStringSet (ModStringSet), 11
- ModDNAStringSet-class (ModStringSet), 11
- ModDNAStringSetList (ModStringSetList), 14
- ModDNAStringSetList-class (ModStringSetList), 14
- modifyNucleotides, 5, 6, 6, 9, 16, 21
- modifyNucleotides, DNAString-method (modifyNucleotides), 6
- modifyNucleotides, DNAStringSet-method (modifyNucleotides), 6
- modifyNucleotides, ModString-method (modifyNucleotides), 6
- modifyNucleotides, ModStringSet-method (modifyNucleotides), 6
- modifyNucleotides, RNAString-method (modifyNucleotides), 6
- modifyNucleotides, RNAStringSet-method (modifyNucleotides), 6
- ModRNAString, 8, 9
- ModRNAString-class (ModString), 9
- ModRNAStringSet (ModStringSet), 11
- ModRNAStringSet-class (ModStringSet), 11
- ModRNAStringSetList (ModStringSetList), 14
- ModRNAStringSetList-class (ModStringSetList), 14
- modsDNA (Modstrings-internals), 10
- modsRNA (Modstrings-internals), 10
- ModString, 3, 5, 7-9, 9, 11, 16, 17
- ModString, AsIs-method (ModString), 9
- ModString, character-method (ModString), 9
- ModString, factor-method (ModString), 9
- ModString, MaskedModString-method (ModString), 9
- ModString, ModString-method (ModString), 9
- ModString, XString-method (ModString), 9
- Modstrings, 10
- Modstrings-internals, 10
- ModStringSet, 3, 7, 8, 10, 11, 13, 14, 16, 17
- ModStringSet, ANY-method (ModStringSet), 11
- ModStringSet, AsIs-method (ModStringSet), 11
- ModStringSet, character-method (ModStringSet), 11
- ModStringSet, factor-method (ModStringSet), 11
- ModStringSet, list-method (ModStringSet), 11
- ModStringSet, missing-method (ModStringSet), 11
- ModStringSet, ModString-method (ModStringSet), 11
- ModStringSet, ModStringSet-method (ModStringSet), 11
- ModStringSet, ModStringViews-method (ModStringViews), 14
- ModStringSet-io, 12
- ModStringSetList, 14
- ModStringViews, 3, 14
- nomenclature (shortName), 21
- nomenclature, ModString-method (shortName), 21
- nomenclature, ModStringSet-method (shortName), 21
- QualityScaledModDNAStringSet, 16
- QualityScaledModDNAStringSet (QualityScaledModStringSet), 15
- QualityScaledModDNAStringSet-class (QualityScaledModStringSet), 15
- QualityScaledModRNAStringSet, 16
- QualityScaledModRNAStringSet (QualityScaledModStringSet), 15
- QualityScaledModRNAStringSet-class (QualityScaledModStringSet), 15
- QualityScaledModStringSet, 15, 20
- readModDNAStringSet (ModStringSet-io), 12



readModRNAStringSet (ModStringSet-io),  
     12  
 readQualityScaledModDNAStringSet  
     (QualityScaledModStringSet), 15  
 readQualityScaledModRNAStringSet  
     (QualityScaledModStringSet), 15  
 removeIncompatibleModifications  
     (separate), 18  
 removeIncompatibleModifications, GRanges, XString-method  
     (separate), 18  
 removeIncompatibleModifications, GRanges, XStringSet-method  
     (separate), 18  
 removeIncompatibleModifications, GRangesList, XStringSet-method  
     (separate), 18  
 replaceLetterAt, 7, 8, 16, 17  
 replaceLetterAt, ModString-method  
     (replaceLetterAt), 16  
 replaceLetterAt, ModStringSet-method  
     (replaceLetterAt), 16  
  
 sanitizeFromModomics (sanitizeInput), 17  
 sanitizeFromtRNADB (sanitizeInput), 17  
 sanitizeInput, 17  
 separate, 18  
 separate, GRanges-method (separate), 18  
 separate, GRangesList-method (separate),  
     18  
 separate, ModString-method (separate), 18  
 separate, ModStringSet-method  
     (separate), 18  
 seqtype, MaskedModString-method  
     (MaskedModString), 4  
 seqtype, ModDNAString-method  
     (Modstrings-internals), 10  
 seqtype, ModRNAString-method  
     (Modstrings-internals), 10  
 seqtype<-, ModString-method  
     (Modstrings-internals), 10  
 seqtype<-, ModStringSet-method  
     (Modstrings-internals), 10  
 shortName, 21  
 shortName, ModString-method (shortName),  
     21  
 shortName, ModStringSet-method  
     (shortName), 21  
 show, ModStringSet-method  
     (ModStringSet), 11  
 show, ModStringViews-method  
     (ModStringViews), 14  
 show, QualityScaledModStringSet-method  
     (QualityScaledModStringSet), 15  
  
 Views, ModString-method  
     (ModStringViews), 14  
 writeModStringSet (ModStringSet-io), 12  
 writeQualityScaledModStringSet  
     (QualityScaledModStringSet), 15  
  
 XString, 5, 9  
 XString, ModString-method  
     (Modstrings-internals), 10  
 XStringQuality, 16  
 XStringSet, 11  
 XStringSet, ModStringSet-method  
     (Modstrings-internals), 10  
 XStringViews, 14, 15