

Package ‘RNAmodR’

September 4, 2025

Type Package

Title Detection of post-transcriptional modifications in high throughput sequencing data

Version 1.23.1

Date 2025-07-22

Description RNAmodR provides classes and workflows for loading/aggregation data from high throughput sequencing aimed at detecting post-transcriptional modifications through analysis of specific patterns. In addition, utilities are provided to validate and visualize the results. The RNAmodR package provides a core functionality from which specific analysis strategies can be easily implemented as a separate package.

biocViews Software, Infrastructure, WorkflowStep, Visualization, Sequencing

License Artistic-2.0

Encoding UTF-8

LazyData false

Depends R (>= 4.0), S4Vectors (>= 0.27.12), IRanges (>= 2.23.9), GenomicRanges, Modstrings

Imports methods, stats, grDevices, matrixStats, BiocGenerics, Biostrings (>= 2.57.2), BiocParallel, txdbmaker, GenomicFeatures, GenomicAlignments, Seqinfo, rtracklayer, Rsamtools, BSgenome, RColorBrewer, colorRamps, ggplot2, Gviz (>= 1.31.0), reshape2, graphics, ROCR

Suggests BiocStyle, knitr, rmarkdown, testthat, RNAmodR.Data

VignetteBuilder knitr

Collate 'RNAmodR.R' 'AllGenerics.R'
'Gviz-ModifiedSequenceTrack-class.R' 'settings.R'
'Modifier-utils.R' 'SequenceDataFrame-class.R'
'normalization.R' 'SequenceData-class.R'
'SequenceDataSet-class.R' 'SequenceDataList-class.R'
'Modifier-class.R' 'ModifierSet-class.R'
'Modifier-Inosine-class.R' 'Modifier-Inosine-viz.R'
'Modifier-roc.R' 'Modifier-subset.R' 'Modifier-viz.R'
'ModifierSet-comparison.R' 'ModifierSet-viz.R'
'RNAmodR-external-functions.R' 'RNAmodR-summary.R'
'SequenceData-coverage.R' 'SequenceData-end-pos.R'

'SequenceData-normalized-end-pos.R' 'SequenceData-pileup.R'
 'SequenceData-viz.R' 'SequenceData-protected-end-pos.R'
 'SequenceData-stats.R' 'SequenceData-subset.R'
 'SequenceDataList-viz.R' 'SequenceDataSet-viz.R' 'utils-bam.R'
 'utils.R'

RoxygenNote 7.3.2

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

URL <https://github.com/FelixErnst/RNAmoR>

git_url <https://git.bioconductor.org/packages/RNAmoR>

git_branch devel

git_last_commit fa3e858

git_last_commit_date 2025-07-22

Repository Bioconductor 3.22

Date/Publication 2025-09-03

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

RNAmoR-package	3
aggregate	3
compare	5
CoverageSequenceData-class	7
EndSequenceData-class	8
Modifier-class	10
Modifier-functions	13
ModifierSet-class	16
modify	18
ModInosine	19
ModInosine-functions	21
ModInosine-internals	22
NormEndSequenceData-class	22
PileupSequenceData-class	24
plotData	25
plotROC	28
ProtectedEndSequenceData-class	30
RNAmoR	31
RNAmoR-datasets	32
RNAmoR-development	33
RNAmoR-internals	36
SequenceData-class	37
SequenceData-functions	39
SequenceDataFrame-class	43
SequenceDataList-class	44
SequenceDataSet-class	44
SequenceModDNAStrngSetTrack-class	45

SequenceModRNAStringSetTrack-class	46
settings	47
stats	48
subsetByCoord	49

Index	52
--------------	-----------

RNAmodR-package	<i>RNAmodR: Detection of post-transcriptional modifications in high throughput sequencing data</i>
-----------------	--

Description

RNAmodR provides classes and workflows for loading/aggregation data from high throughput sequencing aimed at detecting post-transcriptional modifications through analysis of specific patterns. In addition, utilities are provided to validate and visualize the results. The RNAmodR package provides a core functionality from which specific analysis strategies can be easily implemented as a separate package.

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:

- <https://github.com/FelixErnst/RNAmodR>
- Report bugs at <https://github.com/FelixErnst/RNAmodR/issues>

aggregate	<i>Aggregate data per positions</i>
-----------	-------------------------------------

Description

The aggregate function is defined for each [SequenceData](#) object and can be used directly on a [SequenceData](#) object or indirectly via a [Modifier](#) object.

For the letter the call is redirect to the [SequenceData](#) object, the result summarized as defined for the individual Modifier class and stored in the aggregate slot of the Modifier object. The data is then used for subsequent tasks, such as search for modifications and visualization of the results.

The summarization is implemented in the aggregateData for each type of Modifier class. The stored data from the aggregate slot can be retrieved using the getAggregateData function.

Whether the aggregated data is already present in the aggregate slot can be checked using the hasAggregateData function.

For SequenceDataSet, SequenceDataList and ModifierSet classes wrapper of the aggregate function exist as well.

Usage

```

aggregate(x, ...)

aggregateData(x, ...)

getAggregateData(x)

hasAggregateData(x)

## S4 method for signature 'SequenceData'
aggregate(x, condition = c())

## S4 method for signature 'SequenceData'
aggregateData(x, condition)

## S4 method for signature 'SequenceDataSet'
aggregate(x, condition = "Treated")

## S4 method for signature 'SequenceDataList'
aggregate(x, condition = "Treated")

## S4 method for signature 'Modifier'
aggregate(x, force = FALSE)

## S4 method for signature 'Modifier'
aggregateData(x)

## S4 method for signature 'Modifier'
getAggregateData(x)

## S4 method for signature 'Modifier'
hasAggregateData(x)

## S4 method for signature 'ModifierSet'
aggregate(x, force = FALSE)

```

Arguments

x	a SequenceData , SequenceDataSet , SequenceDataList , Modifier or ModifierSet object.
...	additional arguments
condition	character value, which selects, for which condition the data should be aggregated. One of the following values: Both, Control, Treated
force	whether to recreate the aggregated data, if it is already stored inside the Modifier object.

Value

- `aggregate`: for [SequenceData](#) object the aggregated data is returned as a [SplitDataFrameList](#) with an element per transcript, whereas for a [Modifier](#) the modified input object is returned, containing the aggregated data, which can be accessed using `getAggregateData`.

- `getAggregateData`: only for `Modifier`: a `SplitDataFrameList` with an element per transcript is returned. If the aggregated data is not stored in the object, it is generated on the fly, but does not persist.
- `hasAggregateData`: `TRUE` or `FALSE`. Does the `Modifier` object already contain aggregated data?

If 'x' is a

- `SequenceData` a `SplitDataFrameList` with elements per transcript.
- `SequenceDataSet` or `SequenceDataList` a `SimpleList` with `SplitDataFrameList` as elements.
- `Modifier` or `ModifierSet` an updated `Modifier` object. The data can be accessed by using the `aggregateData` function.

Examples

```
data(e5sd, package="RNAmodR")
data(msi, package="RNAmodR")
# modify() triggers the search for modifications in the data contained in
# the Modifier or ModifierSet object
sdf1 <- aggregate(e5sd)
mi <- aggregate(msi[[1]])
```

compare

Comparison of Samples

Description

To compare data of different samples, a `ModifierSet` can be used. To select the data alongside the transcripts and their positions a `GRanges` or a `GRangesList` needs to be provided. In case of a `GRanges` object, the parent column must match the transcript names as defined by the output of `ranges(x)`, whereas in case of a `GRangesList` the element names must match the transcript names.

Usage

```
compare(x, name, pos = 1L, ...)

compareByCoord(x, coord, ...)

plotCompare(x, name, pos = 1L, normalize, ...)

plotCompareByCoord(x, coord, normalize, ...)

## S4 method for signature 'ModifierSet'
compare(x, name, pos = 1L, normalize, ...)

## S4 method for signature 'ModifierSet,GRanges'
compareByCoord(x, coord, normalize, ...)

## S4 method for signature 'ModifierSet,GRangesList'
compareByCoord(x, coord, normalize, ...)
```

```
## S4 method for signature 'ModifierSet'
plotCompare(x, name, pos = 1L, normalize, ...)

## S4 method for signature 'ModifierSet,GRanges'
plotCompareByCoord(x, coord, normalize, ...)

## S4 method for signature 'ModifierSet,GRangesList'
plotCompareByCoord(x, coord, normalize, ...)
```

Arguments

<code>x</code>	a <code>Modifier</code> or <code>ModifierSet</code> object.
<code>name</code>	Only for <code>compare</code> : the transcript name
<code>pos</code>	Only for <code>compare</code> : pos for comparison
<code>...</code>	optional parameters: • <code>alias</code> a <code>data.frame</code> with two columns, <code>tx_id</code> and <code>name</code>, to convert transcript ids to another identifier • <code>name</code> Limit results to one specific gene or transcript • <code>sequenceData</code> <code>TRUE</code> or <code>FALSE</code>? Should the aggregate of <code>sequenceData</code> be used for the comparison instead of the aggregate data if each <code>Modifier</code> element? (default: <code>sequenceData = FALSE</code>) • <code>compareType</code> a valid score type to use for the comparison. If <code>sequenceData = FALSE</code> this defaults to <code>mainScore(x)</code>, whereas if <code>sequenceData = TRUE</code> all columns will be used by setting <code>allTypes = TRUE</code>. • <code>allTypes</code> <code>TRUE</code> or <code>FALSE</code>? Should all available score be compared? (default: <code>allTypes = sequenceData</code>) • ... passed on to subsetByCoord
<code>coord</code>	coordinates of position to subset to. Either a <code>GRanges</code> or a <code>GRangesList</code> object. For both types the 'Parent' column is expected to match the transcript name. The <code>GRangesList</code> object is unlisted and only non duplicated entries are retained.
<code>normalize</code>	either a single logical or character value. If it is a character, it must match one of the names in the <code>ModifierSet</code> .

Value

`compareByCoord` returns a [DataFrame](#) and `plotCompareByCoord` returns a `ggplot` object, which can be modified further. The `DataFrame` contains columns per sample as well as the columns names, positions and `mod` incorporated from the `coord` input. If `coord` contains a column `Activity` this is included in the results as well.

Examples

```
data(msi, package="RNAmodR")
# constructing a GRanges object to mark positive positions
mod <- modifications(msi)
coord <- unique(unlist(mod))
coord$score <- NULL
coord$sd <- NULL
# return a DataFrame
compareByCoord(msi, coord)
# plot the comparison as a heatmap
plotCompareByCoord(msi, coord)
```

CoverageSequenceData-class
CoverageSequenceData

Description

CoverageSequenceData implements [SequenceData](#) to contain and aggregate the coverage of reads per position along the transcripts.

CoverageSequenceData contains one column per data file named using the following naming convention `coverage.condition.replicate`.

`aggregate` calculates the mean and sd for samples in the control and treated condition separately.

Usage

```
CoverageSequenceDataFrame(
  df,
  ranges,
  sequence,
  replicate,
  condition,
  bamfiles,
  seqinfo
)

CoverageSequenceData(bamfiles, annotation, sequences, seqinfo, ...)

## S4 method for signature
## 'CoverageSequenceData,BamFileList,GRangesList,XStringSet,ScanBamParam'
getData(x, bamfiles, grl, sequences, param, args)

## S4 method for signature 'CoverageSequenceData'
aggregateData(x, condition = c("Both", "Treated", "Control"))

## S4 method for signature 'CoverageSequenceData'
getDataTrack(x, name, ...)
```

Arguments

`df, ranges, sequence, replicate` inputs for creating a [SequenceDataFrame](#). See [SequenceDataFrame](#).

`condition` For [aggregate](#): condition for which the data should be aggregated.

`bamfiles, annotation, seqinfo, grl, sequences, param, args, ...` See [SequenceData](#)

`x` a [CoverageSequenceData](#)

`name` For `getDataTrack`: a valid transcript name. Must be a name of `ranges(x)`

Value

a [CoverageSequenceData](#) object

Examples

```
# Construction of a CoverageSequenceData object
library(RNAmoR.Data)
library(rtracklayer)
annotation <- GFF3File(RNAmoR.Data.example.man.gff3())
sequences <- RNAmoR.Data.example.man.fasta()
files <- c(treated = RNAmoR.Data.example.wt.1())
csd <- CoverageSequenceData(files, annotation = annotation,
                           sequences = sequences)
```

EndSequenceData-class *End5SequenceData/End3SequenceData/EndSequenceData*

Description

The End5SequenceData/End3SequenceData/EndSequenceData classes aggregate the counts of read ends at each position along a transcript. End5SequenceData/End3SequenceData classes aggregate either the 5'-end or 3'-end, the EndSequenceData aggregates both.

All three classes contain one column per data file named using the following naming convention (end5/end3/end).condition.replicate.

aggregate calculates the mean and sd for samples in the control and treated condition separately.

Usage

```
End5SequenceDataFrame(
  df,
  ranges,
  sequence,
  replicate,
  condition,
  bamfiles,
  seqinfo
)
```

```
End3SequenceDataFrame(
  df,
  ranges,
  sequence,
  replicate,
  condition,
  bamfiles,
  seqinfo
)
```

```
EndSequenceDataFrame(
  df,
  ranges,
  sequence,
  replicate,
  condition,
```



```

    bamfiles,
    seqinfo
)

End5SequenceData(bamfiles, annotation, sequences, seqinfo, ...)

End3SequenceData(bamfiles, annotation, sequences, seqinfo, ...)

EndSequenceData(bamfiles, annotation, sequences, seqinfo, ...)

## S4 method for signature
## 'End5SequenceData,BamFileList,GRangesList,XStringSet,ScanBamParam'
getData(x, bamfiles, grl, sequences, param, args)

## S4 method for signature
## 'End3SequenceData,BamFileList,GRangesList,XStringSet,ScanBamParam'
getData(x, bamfiles, grl, sequences, param, args)

## S4 method for signature
## 'EndSequenceData,BamFileList,GRangesList,XStringSet,ScanBamParam'
getData(x, bamfiles, grl, sequences, param, args)

## S4 method for signature 'End5SequenceData'
aggregateData(x, condition = c("Both", "Treated", "Control"))

## S4 method for signature 'End3SequenceData'
aggregateData(x, condition = c("Both", "Treated", "Control"))

## S4 method for signature 'EndSequenceData'
aggregateData(x, condition = c("Both", "Treated", "Control"))

## S4 method for signature 'EndSequenceData'
getDataTrack(x, name, ...)

## S4 method for signature 'End5SequenceData'
getDataTrack(x, name, ...)

## S4 method for signature 'End3SequenceData'
getDataTrack(x, name, ...)

```

Arguments

df, ranges, sequence, replicate
 inputs for creating a `SequenceDataFrame`. See [SequenceDataFrame](#).

condition
 For [aggregate](#): condition for which the data should be aggregated.

bamfiles, annotation, seqinfo, grl, sequences, param, args, ...
 See [SequenceData](#) and [SequenceData-functions](#)

x
 a `End5SequenceData`, `End3SequenceData` or `EndSequenceData` object

name
 For [getDataTrack](#): a valid transcript name. Must be a name of `ranges(x)`.

Value

a End5SequenceData, a End3SequenceData or a EndSequenceData object

Examples

```
# Construction of a End5SequenceData object
library(RNAmoR.Data)
library(rtracklayer)
annotation <- GFF3File(RNAmoR.Data.example.man.gff3())
sequences <- RNAmoR.Data.example.man.fasta()
files <- c(treated = RNAmoR.Data.example.wt.1())
e5sd <- End5SequenceData(files, annotation = annotation,
                        sequences = sequences)
```

Modifier-class	<i>The Modifier class</i>
----------------	---------------------------

Description

The Modifier class is a virtual class, which provides the central functionality to search for post-transcriptional RNA modification patterns in high throughput sequencing data.

Each subclass has to implement the following functions:

- Slot nucleotide: Either "RNA" or "DNA". For convenience the subclasses RNAModifier and DNAModifier are already available and can be inherited from.
- Function `aggregateData`: used for specific data aggregation
- Function `findMod`: used for specific search for modifications

Optionally the function `settings<-` can be implemented to store additional arguments, which the base class does not recognize.

Modifier objects are constructed centrally by calling `Modifier()` with a `className` matching the specific class to be constructed. This will trigger the immediate analysis, if `find.mod` is not set to FALSE.

Usage

```
Modifier(className, x, annotation, sequences, seqinfo, ...)
```

```
## S4 method for signature 'SequenceData'
```

```
Modifier(
  className,
  x,
  annotation = NULL,
  sequences = NULL,
  seqinfo = NULL,
  ...
)
```

```
## S4 method for signature 'SequenceDataSet'
```

```
Modifier(
```

```

        className,
        x,
        annotation = NULL,
        sequences = NULL,
        seqinfo = NULL,
        ...
    )

## S4 method for signature 'SequenceDataList'
Modifier(
    className,
    x,
    annotation = NULL,
    sequences = NULL,
    seqinfo = NULL,
    ...
)

## S4 method for signature 'character'
Modifier(
    className,
    x,
    annotation = NULL,
    sequences = NULL,
    seqinfo = NULL,
    ...
)

## S4 method for signature 'list'
Modifier(
    className,
    x,
    annotation = NULL,
    sequences = NULL,
    seqinfo = NULL,
    ...
)

## S4 method for signature 'BamFileList'
Modifier(
    className,
    x,
    annotation = NULL,
    sequences = NULL,
    seqinfo = NULL,
    ...
)

```

Arguments

className	The name of the class which should be constructed.
x	the input which can be of the following types

	• <code>SequenceData</code>: a single <code>SequenceData</code> or a list containing only <code>SequenceData</code> objects. The input will just be used to file the data slot of the <code>Modifier</code> and must match the requirements of specific <code>Modifier</code> class. • <code>BamFileList</code>: a named <code>BamFileList</code> • <code>character</code>: a character vector, which must be coercible to a named <code>BamFileList</code> referencing existing bam files. Valid names are <code>control</code> and <code>treated</code> to define conditions and replicates
<code>annotation</code>	annotation data, which must match the information contained in the BAM files. This parameter is only required if <code>x</code> is not a <code>SequenceData</code> object or a list of <code>SequenceData</code> objects.
<code>sequences</code>	sequences matching the target sequences the reads were mapped onto. This must match the information contained in the BAM files. TThis parameter is only required if <code>x</code> is not a <code>SequenceData</code> object or a list of <code>SequenceData</code> objects.
<code>seqinfo</code>	An optional Seqinfo argument or character vector, which can be coerced to one, to subset the sequences to be analyzed on a per chromosome basis.
<code>...</code>	Additional optional parameters: • <code>find.mod</code>: TRUE or FALSE: should the search for for modifications be triggered upon construction? If not the search can be started by calling the <code>modify()</code> function. • additional parameters depending on the specific <code>Modifier</code> class All additional options must be named and will be passed to the settings function and onto the <code>SequenceData</code> objects, if <code>x</code> is not a <code>SequenceData</code> object or a list of <code>SequenceData</code> objects.

Value

a `Modifier` object of type `className`

Slots

<code>nucleotide</code>	a character value, which needs to contain "RNA" or "DNA"
<code>mod</code>	a character value, which needs to contain one or more elements from the alphabet of a ModRNAString or ModDNAString class.
<code>score</code>	the main score identifier used for visualizations
<code>dataType</code>	the class name(s) of the <code>SequenceData</code> class used
<code>bamfiles</code>	the input bam files as <code>BamFileList</code>
<code>condition</code>	conditions along the <code>BamFileList</code> : Either <code>control</code> or <code>treated</code>
<code>replicate</code>	replicate number along the <code>BamFileList</code> for each of the condition types.
<code>data</code>	The sequence data object: Either a <code>SequenceData</code> , <code>SequenceDataSet</code> or a <code>SequenceDataList</code> object, if more than one <code>dataType</code> is used.
<code>aggregate</code>	the aggregated data as a <code>SplitDataFrameList</code>
<code>modifications</code>	the found modifications as a <code>GRanges</code> object
<code>settings</code>	arguments used for the analysis as a list
<code>aggregateValidForCurrentArguments</code>	TRUE or FALSE whether the aggregate data was constructed with the current arguments
<code>modificationsValidForCurrentArguments</code>	TRUE or FALSE whether the modifications were found with the current arguments

Creation

Modifier objects can be created in two ways, either by providing a list of bamfiles or SequenceData/SequenceDataSet/SequenceDataList objects, which match the structure in dataType().

dataType() can be a character vector or a list of character vectors and depending on this the input files have to follow this structure:

- a single character: a SequenceData is constructed/expected.
- a character vector: a SequenceDataSet is constructed/expected.
- a list of character vectors: a SequenceDataList is constructed/expected.

The cases for a SequenceData or SequenceDataSet are straight forward, since the input remains the same. The last case is special, since it is a hypothetical option, in which bam files from two or more different methods have to be combined to reliably detect a single modification (The elements of a SequenceDataList don't have to be created from the bamfiles, whereas from a SequenceDataSet they have to be).

For this example a list of character vectors is expected. Each element must be named according to the names of dataType() and contain a character vector for creating a SequenceData object.

All additional options must be named and will be passed to the `settings` function and onto the SequenceData objects, if x is not a SequenceData object or a list of SequenceData objects.

Modifier-functions	<i>Modifier/ModifierSet functions</i>
--------------------	---------------------------------------

Description

For the Modifier and ModifierSet classes a number of functions are implemented to access the data stored by the object.

The validAggregate and validModification functions check if `settings` have been modified, after the data was loaded. This potentially invalidates them. To update the data, run the aggregate or the modify function.

Usage

bamfiles(x)

mainScore(x)

modifierType(x)

modType(x)

dataType(x)

sequenceData(x)

sequences(x, ...)

validAggregate(x)

```
validModification(x)

## S4 method for signature 'Modifier'
show(object)

## S4 method for signature 'Modifier'
bamfiles(x)

## S4 method for signature 'Modifier'
conditions(object)

## S4 method for signature 'Modifier'
mainScore(x)

## S4 method for signature 'Modifier'
modifierType(x)

## S4 method for signature 'Modifier'
modType(x)

## S4 method for signature 'Modifier'
dataType(x)

## S4 method for signature 'Modifier'
names(x)

## S4 method for signature 'Modifier'
ranges(x)

## S4 method for signature 'Modifier'
replicates(x)

## S4 method for signature 'Modifier'
seqinfo(x)

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

## S4 method for signature 'Modifier'
sequenceData(x)

## S4 method for signature 'Modifier'
sequences(x, modified = FALSE)

## S4 method for signature 'Modifier'
validAggregate(x)

## S4 method for signature 'Modifier'
validModification(x)

## S4 method for signature 'ModifierSet'
show(object)
```

```

## S4 method for signature 'ModifierSet'
bamfiles(x)

## S4 method for signature 'ModifierSet'
conditions(object)

## S4 method for signature 'ModifierSet'
mainScore(x)

## S4 method for signature 'ModifierSet'
modifications(x, perTranscript = FALSE)

## S4 method for signature 'ModifierSet'
modifierType(x)

## S4 method for signature 'ModifierSet'
modType(x)

## S4 method for signature 'ModifierSet'
dataType(x)

## S4 method for signature 'ModifierSet'
ranges(x)

## S4 method for signature 'ModifierSet'
replicates(x)

## S4 method for signature 'ModifierSet'
seqinfo(x)

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

## S4 method for signature 'ModifierSet'
sequences(x, modified = FALSE)

```

Arguments

<code>x, object</code>	a <code>Modifier</code> or <code>ModifierSet</code> class
<code>...</code>	Additional arguments.
<code>modified</code>	For sequences: TRUE or FALSE: Should the sequences be returned as a <code>ModRNAString</code> / <code>ModDNAString</code> with the found modifications added on top of the <code>RNAString</code> / <code>DNAString</code> ? See combineIntoModstrings .
<code>perTranscript</code>	TRUE or FALSE: Should the positions shown per transcript? (default: <code>perTranscript = FALSE</code>)

Value

- `modifierType`: a character vector with the appropriate class Name of a [Modifier](#).
- `modType`: a character vector with the modifications detected by the `Modifier` class.

- seqtype: a single character value defining if either "RNA" or "DNA" modifications are detected by the Modifier class.
- mainScore: a character vector.
- sequenceData: a SequenceData object.
- modifications: a GRanges or GRangesList object describing the found modifications.
- seqinfo: a Seqinfo object.
- sequences: a RNASTingSet object.
- ranges: a GRangesList object with each element per transcript.
- bamfiles: a BamFileList object.
- validAggregate: TRUE or FALSE. Checks if current settings are the same for which the data was aggregate
- validModification: TRUE or FALSE. Checks if current settings are the same for which modification were found

See Also

[settings](#)

Examples

```
data(msi,package="RNAmodR")
mi <- msi[[1]]
modifierType(mi) # The class name of the Modifier object
modifierType(msi)
seqtype(mi)
modType(mi)
mainScore(mi)
sequenceData(mi)
modifications(mi)
# general accessors
seqinfo(mi)
sequences(mi)
ranges(mi)
bamfiles(mi)
```

ModifierSet-class

The ModifierSet class

Description

The ModifierSet class allows multiple [Modifier](#) objects to be created from the same annotation and sequence data varying only the bam input files.

In addition the comparison of samples is also done via calling functions on the ModifierSet objects.

The ModifierSet is a virtual class, which derives from the SimpleList class with the slot elementType = "Modifier". The ModifierSet class has to be implemented for each specific analysis.#'

Usage

```
ModifierSet(className, x, annotation, sequences, seqinfo, ...)
```

```
## S4 method for signature 'list'
```

```
ModifierSet(
  className,
  x,
  annotation = NULL,
  sequences = NULL,
  seqinfo = NULL,
  ...
)
```

```
## S4 method for signature 'character'
```

```
ModifierSet(
  className,
  x,
  annotation = NULL,
  sequences = NULL,
  seqinfo = NULL,
  ...
)
```

```
## S4 method for signature 'BamFileList'
```

```
ModifierSet(
  className,
  x,
  annotation = NULL,
  sequences = NULL,
  seqinfo = NULL,
  ...
)
```

```
## S4 method for signature 'Modifier'
```

```
ModifierSet(className, x, annotation, sequences, seqinfo, ...)
```

Arguments

className	The name of the class which should be constructed.
x	the input which can be of the following types • Modifier: a single Modifier or a list containing only Modifier objects. The input will just be used as elements of the ModifierSet • BamFileList: a named BamFileList or a list of named BamFileList • list: a list of one or more types of elements: BamFileList, a named list or named character vector. All elements must be or be coercible to a named BamFileList referencing existing bam files. Valid names are control and treated
annotation	annotation data, which must match the information contained in the BAM files. This parameter is only required, if x is not a Modifier object.
sequences	sequences matching the target sequences the reads were mapped onto. This must match the information contained in the BAM files. This parameter is

only required, if `x` is not a `Modifier` object.

`seqinfo` An optional `Seqinfo` argument or character vector, which can be coerced to one, to subset the sequences to be analyzed on a per chromosome basis.

... Additional optional parameters:

- `internalBP` TRUE or FALSE: should parallelization used internally during creation of each `Modifier` or should the creation of the `Modifier` objects be parallelized? (default: `internalBP` = FALSE). Setting `internalBP` only makes sense, if the `getData` function for `SequenceData` class, the `aggregateData` or the `findMod` function contains parallelized code.

All other arguments will be passed onto the `Modifier` objects.

Value

a `ModifierSet` object of type `className`

Creation

The input files have to be provided as a list of elements. Each element in itself must be valid for the creation of `Modifier` object (Have a look at the man page for more details) and must be named.

modify

Searching for modifications in SequenceData

Description

The `modify` function executes the search for modifications for a `Modifier` class. Usually this is done automatically during construction of a `Modifier` object.

When the `modify` functions is called, the aggregated data is checked for validity for the current settings and the search for modifications is performed using the `findMod`. The results are stored in the modification slot of the `Modifier` object, which is returned by `modify`. The results can be accessed via the `modifications()` function.

`findMod` returns the found modifications as a `GRanges` object and has to be implemented for each individual `Modifier` class.

Usage

```
modifications(x, ...)

modify(x, ...)

findMod(x)

## S4 method for signature 'Modifier'
modifications(x, perTranscript = FALSE)

## S4 method for signature 'Modifier'
modify(x, force = FALSE)

## S4 method for signature 'Modifier'
```

```
findMod(x)

## S4 method for signature 'ModifierSet'
modify(x, force = FALSE)
```

Arguments

x	a Modifier object.
...	additional arguments
perTranscript	For modifications> TRUE or FALSE: Should the coordinates be returned as local per transcript coordinates?
force	force to run aggregate again, if data is already stored in x.

Value

- modify: the updated Modifier object.
- modifications: the modifications found as a GRanges object.

Examples

```
data(msi,package="RNAmodR")
# modify() triggers the search for modifications in the data contained in
# the Modifier or ModifierSet object
mi <- modify(msi[[1]])
```

ModInosine

ModInosine

Description

Inosine can be detected in RNA-Seq data by the conversion of A positions to G. This conversion is detected by ModInosine and used to search for Inosine positions. dataType is "PileupSequenceData".

Only samples labeled with the condition treated are used for this analysis, since the A to G conversion is common feature among the reverse transcriptases usually employed. Let us know, if that is not the case, and the class needs to be modified.

Further information on [Functions](#) of ModInosine.

Usage

```
ModInosine(x, annotation, sequences, seqinfo, ...)

ModSetInosine(x, annotation = NA, sequences = NA, seqinfo = NA, ...)
```

Arguments

x	the input which can be of the different types depending on whether a ModRiboMethSeq or a ModSetRiboMethSeq object is to be constructed. For more information have a look at the documentation of the Modifier and ModifierSet classes.
annotation	annotation data, which must match the information contained in the BAM files. This parameter is only required, if x is not a Modifier object.

sequences	sequences matching the target sequences the reads were mapped onto. This must match the information contained in the BAM files. This parameter is only required, if x is not a Modifier object.
seqinfo	An optional Seqinfo argument or character vector, which can be coerced to one, to subset the sequences to be analyzed on a per chromosome basis.
...	Optional arguments overwriting default values, which are • minCoverage: The minimal coverage at the position as integer value (default: minCoverage = 10L). • minReplicate: minimum number of replicates needed for the analysis (default: minReplicate = 1L). • minScore: minimum score to identify Inosine positions de novo (default: minScore = 0.4).

Details

ModInosine score: the scores for reported Inosine positions are between 0 and 1. They are calculated as the relative amount of called G bases ((G / N)) and only saved for genomic A positions.

Value

a ModInosine or ModSetInosine object

Author(s)

Felix G.M. Ernst [aut]

Examples

```
# construction of ModInosine object
library(RNAmoR.Data)
library(rtracklayer)
annotation <- GFF3File(RNAmoR.Data.example.man.gff3())
sequences <- RNAmoR.Data.example.man.fasta()
files <- c(treated = RNAmoR.Data.example.wt.1())
mi <- ModInosine(files, annotation = annotation, sequences = sequences)
# construction of ModSetInosine object
## Not run:
files <- list("SampleSet1" = c(treated = RNAmoR.Data.example.wt.1(),
                              treated = RNAmoR.Data.example.wt.2(),
                              treated = RNAmoR.Data.example.wt.3()),
             "SampleSet2" = c(treated = RNAmoR.Data.example.bud23.1(),
                              treated = RNAmoR.Data.example.bud23.2()),
             "SampleSet3" = c(treated = RNAmoR.Data.example.trm8.1(),
                              treated = RNAmoR.Data.example.trm8.2()))
msi <- ModSetInosine(files, annotation = annotation, sequences = sequences)

## End(Not run)
```

Description

All of the functions of [Modifier](#) and the [ModifierSet](#) classes are inherited by the `ModInosine` and `ModSetInosine` classes.

Check below for the specifically implemented functions.

Usage

```
## S4 replacement method for signature 'ModInosine'
settings(x) <- value

## S4 method for signature 'ModInosine'
aggregateData(x)

## S4 method for signature 'ModInosine'
findMod(x)

## S4 method for signature 'ModInosine'
getDataTrack(x, name, type, ...)

## S4 method for signature 'ModInosine,GRanges'
plotDataByCoord(x, coord, type = "score", window.size = 15L, ...)

## S4 method for signature 'ModInosine'
plotData(x, name, from = 1L, to = 30L, type = "score", ...)

## S4 method for signature 'ModSetInosine,GRanges'
plotDataByCoord(x, coord, type = "score", window.size = 15L, ...)

## S4 method for signature 'ModSetInosine'
plotData(x, name, from = 1L, to = 30L, type = "score", ...)
```

Arguments

<code>x</code>	a Modifier or a ModifierSet object. For more details see also the man pages for the functions mentioned below.
<code>value</code>	See settings
<code>coord, name, from, to, type, window.size, ...</code>	See plotData

Details

`ModInosine` specific arguments for [plotData](#):

- `colour.bases` - a named character vector of length = 4 for the colours of the individual bases. The names are expected to be `c("G", "A", "U", "C")`

Value

- settings See [settings](#).
- aggregate See [aggregate](#).
- modify See [modify](#).
- getDataTrack a list of [DataTrack](#) objects. See [plotDataByCoord](#).
- plotData See [plotDataByCoord](#).
- plotDataByCoord See [plotDataByCoord](#).

Examples

```
data(msi,package="RNAmodR")
mi <- msi[[1]]
settings(mi)
## Not run:
aggregate(mi)
modify(mi)

## End(Not run)
getDataTrack(mi, "1", mainScore(mi))
```

ModInosine-internals *ModInosine internal functions*

Description

These functions are not intended for general use, but are used for additional package development.

Arguments

x, data, seqdata, sequence, args
internally used arguments

NormEndSequenceData-class
NormEnd5SequenceData/NormEnd3SequenceData

Description

The NormEnd5SequenceData/NormEnd3SequenceData aggregate the counts of read ends (Either 5' or 3') at each position along a transcript. In addition, the number of counts are then normalized to the length of the transcript and to the overlapping reads.

Both classes contain three columns per data file named using the following naming convention (normend5/normend3).condition.replicate. The three columns are distinguished by additional identifiers ends, norm.tx and norm.ol.

aggregate calculates the mean and sd for samples in the control and treated condition separatly. Similar to the stored results for each of the two conditions six columns are returned (three for mean and sd each) ending in ends, tx and ol.

Usage

```

NormEnd5SequenceDataFrame(
  df,
  ranges,
  sequence,
  replicate,
  condition,
  bamfiles,
  seqinfo
)

NormEnd3SequenceDataFrame(
  df,
  ranges,
  sequence,
  replicate,
  condition,
  bamfiles,
  seqinfo
)

NormEnd5SequenceData(bamfiles, annotation, sequences, seqinfo, ...)

NormEnd3SequenceData(bamfiles, annotation, sequences, seqinfo, ...)

## S4 method for signature
## 'NormEnd5SequenceData,BamFileList,GRangesList,XStringSet,ScanBamParam'
getData(x, bamfiles, grl, sequences, param, args)

## S4 method for signature
## 'NormEnd3SequenceData,BamFileList,GRangesList,XStringSet,ScanBamParam'
getData(x, bamfiles, grl, sequences, param, args)

## S4 method for signature 'NormEnd5SequenceData'
aggregateData(x, condition = c("Both", "Treated", "Control"))

## S4 method for signature 'NormEnd3SequenceData'
aggregateData(x, condition = c("Both", "Treated", "Control"))

## S4 method for signature 'NormEnd5SequenceData'
getDataTrack(x, name, ...)

## S4 method for signature 'NormEnd3SequenceData'
getDataTrack(x, name, ...)

```

Arguments

df, ranges, sequence, replicate
 inputs for creating a SequenceDataFrame. See [SequenceDataFrame](#).

condition
 For [aggregate](#): condition for which the data should be aggregated.

bamfiles, annotation, seqinfo, grl, sequences, param, args, ...
 See [SequenceData](#) and [SequenceData-functions](#)

x a CoverageSequenceData
 name For `getDataTrack`: a valid transcript name. Must be a name of ranges(x)

Value

a NormEnd5SequenceData or NormEnd3SequenceData object

Examples

```
# Construction of a NormEnd5SequenceData object
## Not run:
library(RNAmoR.Data)
library(rtracklayer)
annotation <- GFF3File(RNAmoR.Data.example.man.gff3())
sequences <- RNAmoR.Data.example.man.fasta()
files <- c(treated = RNAmoR.Data.example.wt.1())
ne5sd <- NormEnd5SequenceData(files, annotation = annotation,
                             sequences = sequences)

## End(Not run)
```

PileupSequenceData-class

PileupSequenceData

Description

The PileupSequenceData aggregates the pileup of called bases per position.

PileupSequenceData contains five columns per data file named using the following naming convention `pileup.condition.replicate`. The five columns are distinguished by additional identifiers -, G, A, T and C.

`aggregate` calculates the mean and sd for each nucleotide in the control and treated condition separately. The results are then normalized to a row sum of 1.

Usage

```
PileupSequenceDataFrame(
  df,
  ranges,
  sequence,
  replicate,
  condition,
  bamfiles,
  seqinfo
)

PileupSequenceData(bamfiles, annotation, sequences, seqinfo, ...)

## S4 method for signature
## 'PileupSequenceData,BamFileList,GRangesList,XStringSet,ScanBamParam'
getData(x, bamfiles, grl, sequences, param, args)
```



```
## S4 method for signature 'PileupSequenceData'
aggregateData(x, condition = c("Both", "Treated", "Control"))

## S4 method for signature 'PileupSequenceData'
getDataTrack(x, name, ...)

pileupToCoverage(x)

## S4 method for signature 'PileupSequenceData'
pileupToCoverage(x)
```

Arguments

df, ranges, sequence, replicate
 inputs for creating a `SequenceDataFrame`. See [SequenceDataFrame](#).

condition For [aggregate](#): condition for which the data should be aggregated.

bamfiles, annotation, seqinfo, grl, sequences, param, args, ...
 See [SequenceData](#) and [SequenceData-functions](#)

x a `PileupSequenceData`

name For [getDataTrack](#): a valid transcript name. Must be a name of `ranges(x)`

Value

a `PileupSequenceData` object

Examples

```
# Construction of a PileupSequenceData object
library(RNAmoR.Data)
library(rtracklayer)
annotation <- GFF3File(RNAmoR.Data.example.man.gff3())
sequences <- RNAmoR.Data.example.man.fasta()
files <- c(treated = RNAmoR.Data.example.wt.1())
psd <- PileupSequenceData(files, annotation = annotation,
                           sequences = sequences)
```

plotData	<i>Visualizing data data from a <code>SequenceData</code>, <code>SequenceDataSet</code>, <code>SequenceDataList</code>, <code>Modifier</code> or <code>ModifierSet</code> object.</i>
----------	---

Description

With the `plotData` and `plotDataByCoord` functions data from a `SequenceData`, `SequenceDataSet`, `SequenceDataList`, `Modifier` or `ModifierSet` object can be visualized.

Internally the functionality of the `Gviz` package is used. For each `SequenceData` and `Modifier` class the `getDataTrack` is implemented returning a [DataTrack](#) object from the `Gviz` package.

Positions to be visualized are selected by defining a genomic coordinate, for which `x` has to contain data.

Usage

```

plotData(x, name, from = 1L, to = 30L, type, ...)

plotDataByCoord(x, coord, type, window.size = 15L, ...)

getDataTrack(x, name, ...)

## S4 method for signature 'Modifier,GRanges'
plotDataByCoord(x, coord, type = NA, window.size = 15L, ...)

## S4 method for signature 'Modifier'
plotData(
  x,
  name,
  from,
  to,
  type = NA,
  showSequenceData = FALSE,
  showSequence = TRUE,
  showAnnotation = FALSE,
  ...
)

## S4 method for signature 'Modifier'
getDataTrack(x, name = name, ...)

## S4 method for signature 'ModifierSet,GRanges'
plotDataByCoord(x, coord, type = NA, window.size = 15L, ...)

## S4 method for signature 'ModifierSet'
plotData(
  x,
  name,
  from,
  to,
  type = NA,
  showSequenceData = FALSE,
  showSequence = TRUE,
  showAnnotation = FALSE,
  ...
)

## S4 method for signature 'SequenceData,GRanges'
plotDataByCoord(x, coord, type = NA, window.size = 15L, ...)

## S4 method for signature 'SequenceData'
plotData(
  x,
  name,
  from,
  to,
  perTranscript = FALSE,

```

```

        showSequence = TRUE,
        showAnnotation = FALSE,
        ...
    )

    ## S4 method for signature 'SequenceData'
    getDataTrack(x, name = name, ...)

    ## S4 method for signature 'SequenceDataList'
    getDataTrack(x, name = name, ...)

    ## S4 method for signature 'SequenceDataList,GRanges'
    plotDataByCoord(x, coord, type = NA, window.size = 15L, ...)

    ## S4 method for signature 'SequenceDataList'
    plotData(
        x,
        name,
        from,
        to,
        perTranscript = FALSE,
        showSequence = TRUE,
        showAnnotation = FALSE,
        ...
    )

    ## S4 method for signature 'SequenceDataSet'
    getDataTrack(x, name = name, ...)

    ## S4 method for signature 'SequenceDataSet,GRanges'
    plotDataByCoord(x, coord, type = NA, window.size = 15L, ...)

    ## S4 method for signature 'SequenceDataSet'
    plotData(
        x,
        name,
        from,
        to,
        perTranscript = FALSE,
        showSequence = TRUE,
        showAnnotation = FALSE,
        ...
    )

```

Arguments

x	a SequenceData, SequenceDataSet, SequenceDataList, Modifier or ModifierSet object.
name	Only for plotData: the transcript name
from	Only for plotData: start position
to	Only for plotData: end position
type	the data type of data show as data tracks.

...	optional parameters:
	• <code>modified.seq</code> TRUE or FALSE. Should the sequence shown with modified nucleotide positions? (default: <code>modified.seq = FALSE</code>) • <code>additional.mod</code> other modifications, which should be shown in the annotation and sequence track. The must be a GRanges compatible with combineIntoModstrings. • <code>annotation.track.pars</code> Parameters passed onto the AnnotationTrack. • <code>sequence.track.pars</code> Parameters passed onto the SequenceTrack.
<code>coord</code>	coordinates of a positions to subset to as a GRanges object. The 'Parent' column is expected to match the transcript name.
<code>window.size</code>	integer value for the number of positions on the left and right site of the selected positions included in the plotting (default: <code>window.size = 15L</code>)
<code>showSequenceData</code>	TRUE or FALSE: should the sequence data be shown? (default: <code>seqdata = FALSE</code>)
<code>showSequence</code>	TRUE or FALSE: should a sequence track be shown? (default: <code>seqdata = TRUE</code>)
<code>showAnnotation</code>	TRUE or FALSE: should an annotation track be shown? (default: <code>seqdata = FALSE</code>)
<code>perTranscript</code>	TRUE or FALSE: Should the positions shown per transcript? (default: <code>perTranscript = FALSE</code>)

Value

a plot send to the active graphic device

Examples

```
data(msi,package="RNAmodR")
plotData(msi[[1]], "2", from = 10L, to = 45L)
## Not run:
plotData(msi, "2", from = 10L, to = 45L)

## End(Not run)
```

plotROC

ROCR functions for Modifier and ModifierSet objects

Description

plotROC streamlines labeling, prediction, performance and plotting functions to test the performance of a Modifier object and the data analyzed via the functionality from the ROCR package.

The data from `x` will be labeled as positive using the `coord` arguments. The other arguments will be passed on to the specific ROCR functions.

By default the `prediction.args` include three values:

- `measure = "tpr"`
- `x.measure = "fpr"`
- `score = mainScore(x)`

The remaining arguments are not predefined.

Usage

```
plotROC(x, coord, ...)

## S4 method for signature 'Modifier'
plotROC(
  x,
  coord,
  score = NULL,
  prediction.args = list(),
  performance.args = list(),
  plot.args = list()
)

## S4 method for signature 'ModifierSet'
plotROC(
  x,
  coord,
  score = NULL,
  prediction.args = list(),
  performance.args = list(),
  plot.args = list()
)
```

Arguments

<code>x</code>	a <code>Modifier</code> or a <code>ModifierSet</code> object
<code>coord</code>	coordinates of position to label as positive. Either a <code>GRanges</code> or a <code>GRangesList</code> object. For both types the <code>Parent</code> column is expected to match the gene or transcript name.
<code>...</code>	additional arguments
<code>score</code>	the score identifier to subset to, if multiple scores are available.
<code>prediction.args</code>	arguments which will be used for calling <code>prediction</code> form the ROCR package
<code>performance.args</code>	arguments which will be used for calling <code>performance</code> form the ROCR package
<code>plot.args</code>	arguments which will be used for calling <code>plot</code> on the performance object of the ROCR package. If multiple scores are plotted (for example if the <code>score</code> argument is not explicitly set) <code>add = FALSE</code> will be set.

Value

a plot send to the active graphic device

References

Tobias Sing, Oliver Sander, Niko Beerenwinkel, Thomas Lengauer (2005): "ROCR: visualizing classifier performance in R." *Bioinformatics* 21(20):3940-3941 DOI: [10.1093/bioinformatics/bti623](https://doi.org/10.1093/bioinformatics/bti623)

Examples

```
data(msi,package="RNAmodR")
# constructing a GRanges object to mark positive positions
mod <- modifications(msi)
coord <- unique(unlist(mod))
coord$score <- NULL
coord$sd <- NULL
# plotting a TPR vs. FPR plot per ModInosine object
plotROC(msi[[1]],coord)
# plotting a TPR vs. FPR plot per ModSetInosine object
plotROC(msi,coord)
```

ProtectedEndSequenceData-class

ProtectedEndSequenceData

Description

ProtectedEndSequenceData implements [SequenceData](#) to contain and aggregate the start and ends of reads per position along a transcript. ProtectedEndSequenceData offsets the start position by -1 to align the information on the 5'-3'-phosphate bonds to one position. The ProtectedEndSequenceData class is implemented specifically as required for the RiboMethSeq method.

The objects of type ProtectedEndSequenceData contain three columns per data file named using the following naming convention protectedend.condition.replicate.

aggregate calculates the mean and sd for samples in the control and treated condition separately.

Usage

```
ProtectedEndSequenceDataFrame(
  df,
  ranges,
  sequence,
  replicate,
  condition,
  bamfiles,
  seqinfo
)
```

```
ProtectedEndSequenceData(bamfiles, annotation, sequences, seqinfo, ...)
```

```
## S4 method for signature
## 'ProtectedEndSequenceData,
## BamFileList,
## GRangesList,
## XStringSet,
## ScanBamParam'
getData(x, bamfiles, grl, sequences, param, args)
```

```
## S4 method for signature 'ProtectedEndSequenceData'
aggregateData(x, condition = c("Both", "Treated", "Control"))
```

```
## S4 method for signature 'ProtectedEndSequenceData'
getDataTrack(x, name, ...)
```

Arguments

df, ranges, sequence, replicate
 inputs for creating a `SequenceDataFrame`. See [SequenceDataFrame](#).
 condition For [aggregate](#): condition for which the data should be aggregated.
 bamfiles, annotation, seqinfo, grl, sequences, param, args, ...
 See [SequenceData](#) and [SequenceData-functions](#)
 x a `ProtectedEndSequenceData`
 name For [getDataTrack](#): a valid transcript name. Must be a name of `ranges(x)`

Value

a `ProtectedEndSequenceData` object

Examples

```
# Construction of a ProtectedEndSequenceData object
library(RNAmodR.Data)
library(rtracklayer)
annotation <- GFF3File(RNAmodR.Data.example.man.gff3())
sequences <- RNAmodR.Data.example.man.fasta()
files <- c(treated = RNAmodR.Data.example.wt.1())
pesd <- ProtectedEndSequenceData(files, annotation = annotation,
                                sequences = sequences)
```

 RNAmodR

RNAmodR

Description

Post-transcriptional modifications can be found abundantly in rRNA and tRNA and can be detected classically via several strategies. However, difficulties arise if the identity and the position of the modified nucleotides is to be determined at the same time. Classically, a primer extension, a form of reverse transcription (RT), would allow certain modifications to be accessed by blocks during the RT changes or changes in the cDNA sequences. Other modification would need to be selectively treated by chemical reactions to influence the outcome of the reverse transcription.

With the increased availability of high throughput sequencing, these classical methods were adapted to high throughput methods allowing more RNA molecules to be accessed at the same time. With these advances post-transcriptional modifications were also detected on mRNA. Among these high throughput techniques are for example Pseudo-Seq (Carlile et al. 2014), RiboMethSeq (Birkedal et al. 2015) and AlkAnilineSeq (Marchand et al. 2018) each able to detect a specific type of modification from footprints in RNA-Seq data prepared with the selected methods.

Since similar pattern can be observed from some of these techniques, overlaps of the bioinformatical pipeline already are and will become more frequent with new emerging sequencing techniques.

RNAmodR implements classes and a workflow to detect post-transcriptional RNA modifications in high throughput sequencing data. It is easily adaptable to new methods and can help during the phase of initial method development as well as more complex screenings.

Briefly, from the SequenceData, specific subclasses are derived for accessing specific aspects of aligned reads, e.g. 5'-end positions or pileup data. With this a Modifier class can be used to detect specific patterns for individual types of modifications. The SequenceData classes can be shared by different Modifier classes allowing easy adaptation to new methods.

Author(s)

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

References

- Carlile TM, Rojas-Duran MF, Zinshteyn B, Shin H, Bartoli KM, Gilbert WV (2014): "Pseudouridine profiling reveals regulated mRNA pseudouridylation in yeast and human cells." *Nature* 515 (7525), P. 143–146. DOI: [10.1038/nature13802](https://doi.org/10.1038/nature13802).
- Birkedal U, Christensen-Dalsgaard M, Krogh N, Sabarinathan R, Gorodkin J, Nielsen H (2015): "Profiling of ribose methylations in RNA by high-throughput sequencing." *Angewandte Chemie (International ed. in English)* 54 (2), P. 451–455. DOI: [10.1002/anie.201408362](https://doi.org/10.1002/anie.201408362).
- Marchand V, Ayadi L, __Ernst FGM__, Hertler J, Bourguignon-Igel V, Galvanin A, Kotter A, Helm M, __Lafontaine DLJ__, Motorin Y (2018): "AlkAniline-Seq: Profiling of m7 G and m3 C RNA Modifications at Single Nucleotide Resolution." *Angewandte Chemie (International ed. in English)* 57 (51), P. 16785–16790. DOI: [10.1002/anie.201810946](https://doi.org/10.1002/anie.201810946).

See Also

The RNAmodR.RiboMethSeq and RNAmodR.AlkAnilineSeq package.

RNAmodR-datasets

Example data in the RNAmodR package

Description

The following datasets are contained in the RNAmodR package. They are used in the man page examples.

Usage

`data(msi)`

`data(sds)`

`data(sdl)`

`data(psd)`

`data(e5sd)`

`data(e3sd)`

`data(esd)`

`data(csd)`


```
data(ne3sd)
```

```
data(ne5sd)
```

```
data(pesd)
```

Format

- msi a ModSetInosine instance
- sds a SequenceDataSet instance
- sdl a SequenceDataList instance
- psd a PileupSequenceData instance
- e5sd a End5SequenceData instance
- e3sd a End3SequenceData instance
- esd a EndSequenceData instance
- csd a CoverageSequenceData instance
- ne3sd a NormEnd3SequenceData instance
- ne5sd a NormEnd5SequenceData instance
- pesd a ProtectedEndSequenceData instance

An object of class SequenceDataSet of length 2.

An object of class SequenceDataList of length 3.

An object of class PileupSequenceData of dimension 100 x 101 x 15 x 15.

An object of class End5SequenceData of dimension 100 x 101 x 3 x 3.

An object of class End3SequenceData of dimension 100 x 101 x 3 x 3.

An object of class EndSequenceData of dimension 100 x 101 x 3 x 3.

An object of class CoverageSequenceData of dimension 100 x 101 x 3 x 3.

An object of class NormEnd3SequenceData of dimension 100 x 101 x 9 x 9.

An object of class NormEnd5SequenceData of dimension 100 x 101 x 9 x 9.

An object of class ProtectedEndSequenceData of dimension 100 x 101 x 3 x 3.

RNAmodR-development *RNAmodR developments functions*

Description

These functions are not intended for general use, but are used for additional package development.

getData is used to load data into a [SequenceData](#) object and must be implemented for all SequenceData classes. The results must match the requirements outlined in the value section.

In addition the following functions should be implemented for complete functionality:

aggregateData for each SequenceData and Modifier class. See also [aggregateData](#)

findMod for each Modifier class. See also [findMod](#).

plotData/plotDataByCoord for each Modifier and ModifierSet class. See also [plotData](#).

The following helper function can be called from within `findMod` to construct a coordinate for each modification found:

`constructModRanges` constructs a `GRanges` object describing the location, type and associated scores of a modification. `constructModRanges` is typically called from the `modify` function, which must be implemented for all `Modifier` classes.

Usage

```
constructModRanges(range, data, modType, scoreFun, source, type)
```

```
getData(x, bamfiles, grl, sequences, param, args)
```

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

```
constructModRanges(range, data, modType, scoreFun, source, type)
```

Arguments

<code>range</code>	for <code>constructModRanges</code> : a <code>GRanges</code> object
<code>data</code>	for <code>constructModRanges</code> : a <code>DataFrame</code> object
<code>modType</code>	for <code>constructModRanges</code> : a valid <code>shortName</code> for the modification found. Must be present in <code>shortName(ModRNAString())</code> .
<code>scoreFun</code>	for <code>constructModRanges</code> : a custom function for extracting scores from data. The result must be a list.
<code>source</code>	for <code>constructModRanges</code> : a single character vector for populating the source column of the result.
<code>type</code>	for <code>constructModRanges</code> : a single character vector for populating the source column of the result.
<code>x</code>	for <code>getData</code> : a <code>SequenceData</code> object.
<code>bamfiles</code>	for <code>getData</code> : a <code>BamFileList</code> object.
<code>grl</code>	for <code>getData</code> : a <code>GRangesList</code> object.
<code>sequences</code>	for <code>getData</code> : a <code>XStringSet</code> object.
<code>param</code>	for <code>getData</code> : a <code>ScanBamParam</code> object.
<code>args</code>	for <code>getData</code> : a list with optional arguments.

Value

- `getData`: returns a list with elements per `BamFile` in `bamfiles`. Elements can be `IntegerList`, `NumericList` or a `CompressedSplitDataFrameList`. The data in the elements must be order by increasing positions numbers. However, names and rownames will be discarded.
- `constructModRanges`: returns a `GRanges` object with genomic coordinates of modified nucleotides in the associated transcripts.

Examples

```
# new SequenceData class
setClass(Class = "ExampleSequenceData",
         contains = "SequenceData",
         prototype = list(minQuality = 5L))
ExampleSequenceData <- function(bamfiles, annotation, sequences, seqinfo, ...){
  RNAmodR::SequenceData("Example", bamfiles = bamfiles,
```

```

        annotation = annotation, sequences = sequences,
        seqinfo = seqinfo, ...)
    }
    setMethod("getData",
              signature = c(x = "ExampleSequenceData",
                           bamfiles = "BamFileList",
                           grl = "GRangesList",
                           sequences = "XStringSet",
                           param = "ScanBamParam"),
              definition = function(x, bamfiles, grl, sequences, param, args){
                ###
              }
    )
    setMethod("aggregateData",
              signature = c(x = "ExampleSequenceData"),
              function(x, condition = c("Both", "Treated", "Control")){
                ###
              }
    )
    setMethod(
      f = "getDataTrack",
      signature = c(x = "ExampleSequenceData"),
      definition = function(x, name, ...) {
        ###
      }
    )
    # new Modifier class
    setClass("ModExample",
             contains = "Modifier",
             prototype = list(mod = "X",
                              score = "score",
                              dataType = "ExampleSequenceData"))
    ModExample <- function(x, annotation, sequences, seqinfo, ...){
      RNAmodR::Modifier("ModExample", x = x, annotation = annotation,
                        sequences = sequences, seqinfo = seqinfo, ...)
    }

    setMethod(f = "aggregateData",
              signature = c(x = "ModExample"),
              definition =
                function(x, force = FALSE){
                  # Some data with element per transcript
                }
    )

    setMethod("findMod",
              signature = c(x = "ModExample"),
              function(x){
                # an element per modification found.
              }
    )
    setMethod(
      f = "getDataTrack",
      signature = signature(x = "ModExample"),
      definition = function(x, name, type, ...) {
      }
    )

```

```

)
setMethod(
  f = "plotDataByCoord",
  signature = signature(x = "ModExample", coord = "GRanges"),
  definition = function(x, coord, type = "score", window.size = 15L, ...) {
  }
)
setMethod(
  f = "plotData",
  signature = signature(x = "ModExample"),
  definition = function(x, name, from, to, type = "score", ...) {
  }
)

# new ModifierSet class
setClass("ModSetExample",
  contains = "ModifierSet",
  prototype = list(elementType = "ModExample"))
ModSetExample <- function(x, annotation, sequences, seqinfo, ...){
  RNAmodR::ModifierSet("ModExample", x = x, annotation = annotation,
    sequences = sequences, seqinfo = seqinfo, ...)
}

setMethod(
  f = "plotDataByCoord",
  signature = signature(x = "ModSetExample", coord = "GRanges"),
  definition = function(x, coord, type = "score", window.size = 15L, ...) {
  }
)
setMethod(
  f = "plotData",
  signature = signature(x = "ModSetExample"),
  definition = function(x, name, from, to, type = "score", ...) {
  }
)

```

RNAmodR-internals

RNAmodR internal functions

Description

These functions are used internally.

Usage

```

## S4 method for signature 'SequenceDataSet'
parallel_slot_names(x)

## S4 method for signature 'SequenceDataSet'
getListElement(x, i, exact = TRUE)

## S4 method for signature 'SequenceDataList'
parallel_slot_names(x)

```

```
## S4 method for signature 'SequenceDataList'
getListElement(x, i, exact = TRUE)

## S4 method for signature 'Modifier'
subset(x, name, pos = 1L, ...)

## S4 method for signature 'SplitDataFrameList,GRanges'
labelByCoord(x, coord, ...)
```

Arguments

object, range, data, modType, scoreFun, source, type, x, i, j, ..., exact,
value

Internal arguments

Value

internally used values

SequenceData-class	<i>The SequenceData class</i>
--------------------	-------------------------------

Description

The SequenceData class is implemented to contain data on each position along transcripts and holds the corresponding annotation data and nucleotide sequence of these transcripts. To access this data several [functions](#) are available. The SequenceData class is a virtual class, from which specific classes can be extended. Currently the following classes are implemented:

- [CoverageSequenceData](#)
- [End5SequenceData](#), [End3SequenceData](#), [EndSequenceData](#)
- [NormEnd5SequenceData](#), [NormEnd5SequenceData](#)
- [PileupSequenceData](#)
- [ProtectedEndSequenceData](#)

The annotation and sequence data can be accessed through the functions `ranges` and `sequences`, respectively. Be aware, that the data is always provided according to genomic positions with increasing rownames, but the sequence is given as the actual sequence of the transcript. Therefore, it is necessary to treat the minus strand accordingly.

The SequenceData class is derived from the [CompressedSplitDataFrameList](#) class with additional slots for annotation and sequence data. Some functionality is not inherited and might not be available to full extend, e.g. `relist`.

SequenceDataFrame

The SequenceDataFrame class is a virtual class and contains data for positions along a single transcript. In addition to being used for returning elements from a SequenceData object, the SequenceDataFrame class is used to store the unlisted data within a [SequenceData](#) object. Therefore, a matching SequenceData and SequenceDataFrame class must be implemented.

The SequenceDataFrame class is derived from the [DataFrame](#) class.

Subsetting of a SequenceDataFrame returns a SequenceDataFrame or DataFrame, if it is subset by a column or row, respectively. The `drop` argument is ignored for column subsetting.

Usage

```
## S4 method for signature 'SequenceData'
cbind(..., deparse.level = 1)

## S4 method for signature 'SequenceData'
rbind(..., deparse.level = 1)

SequenceData(dataType, bamfiles, annotation, sequences, seqinfo, ...)

## S4 method for signature 'character,character'
SequenceData(dataType, bamfiles, annotation, sequences, seqinfo, ...)

## S4 method for signature 'character,BSgenome'
SequenceData(dataType, bamfiles, annotation, sequences, seqinfo, ...)

## S4 method for signature 'TxDb,character'
SequenceData(dataType, bamfiles, annotation, sequences, seqinfo, ...)

## S4 method for signature 'TxDb,BSgenome'
SequenceData(dataType, bamfiles, annotation, sequences, seqinfo, ...)

## S4 method for signature 'GRangesList,character'
SequenceData(dataType, bamfiles, annotation, sequences, seqinfo, ...)

## S4 method for signature 'GRangesList,BSgenome'
SequenceData(dataType, bamfiles, annotation, sequences, seqinfo, ...)

## S4 method for signature 'GFF3File,BSgenome'
SequenceData(dataType, bamfiles, annotation, sequences, seqinfo, ...)

## S4 method for signature 'GFF3File,character'
SequenceData(dataType, bamfiles, annotation, sequences, seqinfo, ...)

## S4 method for signature 'character,FaFile'
SequenceData(dataType, bamfiles, annotation, sequences, seqinfo, ...)

## S4 method for signature 'GFF3File,FaFile'
SequenceData(dataType, bamfiles, annotation, sequences, seqinfo, ...)

## S4 method for signature 'TxDb,FaFile'
SequenceData(dataType, bamfiles, annotation, sequences, seqinfo, ...)

## S4 method for signature 'GRangesList,FaFile'
SequenceData(dataType, bamfiles, annotation, sequences, seqinfo, ...)
```

Arguments

- ...
- Optional arguments overwriting default values. Not all SequenceData classes use all arguments. The arguments are:
- `minLength` single integer value setting a threshold for minimum read length. Shorter reads are discarded (default: `minLength = NA`).

	• <code>maxLength</code> single integer value setting a threshold for maximum read length. Longer reads are discarded (default: <code>maxLength = NA</code>). • <code>minQuality</code> single integer value setting a threshold for maximum read quality. Reads with a lower quality are discarded (default: <code>minQuality = 5L</code>, but this is class dependent). • <code>max_depth</code> maximum depth for pileup loading (default: <code>max_depth = 10000L</code>).
<code>deparse.level</code>	See base::cbind for a description of this argument.
<code>dataType</code>	The prefix for construction the class name of the <code>SequenceData</code> subclass to be constructed.
<code>bamfiles</code>	the input which can be of the following types • <code>BamFileList</code>: a named <code>BamFileList</code> • <code>character</code>: a character vector, which must be coercible to a named <code>BamFileList</code> referencing existing bam files. Valid names are <code>control</code> and <code>treated</code> to define conditions and replicates
<code>annotation</code>	annotation data, which must match the information contained in the BAM files.
<code>sequences</code>	sequences matching the target sequences the reads were mapped onto. This must match the information contained in the BAM files.
<code>seqinfo</code>	optional Seqinfo to subset the transcripts analyzed on a chromosome basis.

Value

A `SequenceData` object

Slots

`sequencesType` a character value for the class name of sequences. Either `RNAStringSet`, `ModRNAStringSet`, `DNAStringSet` or `ModDNAStringSet`.

`minQuality` a integer value describing a threshold of the minimum quality of reads to be used.

SequenceData-functions

*SequenceData/SequenceDataSet/SequenceDataList/SequenceDataFrame
functions*

Description

The `SequenceData`, `SequenceDataSet`, `SequenceDataList` and `SequenceDataFrame` classes share functionality. Have a look at the elements listed directly below.

Usage

```
replicates(x)
```

```
## S4 method for signature 'SequenceDataFrame'
show(object)
```

```
## S4 method for signature 'SequenceDataFrame'
conditions(object)
```

```
## S4 method for signature 'SequenceDataFrame'
bamfiles(x)

## S4 method for signature 'SequenceDataFrame'
dataType(x)

## S4 method for signature 'SequenceDataFrame'
ranges(x)

## S4 method for signature 'SequenceDataFrame'
replicates(x)

## S4 method for signature 'SequenceDataFrame'
seqinfo(x)

## S4 method for signature 'SequenceDataFrame'
seqinfo(x)

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

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

## S4 method for signature 'SequenceDataFrame'
sequences(x)

## S4 method for signature 'SequenceData'
show(object)

## S4 method for signature
## 'SequenceData,BamFileList,GRangesList,XStringSet,ScanBamParam'
getData(x, bamfiles, grl, sequences, param, args)

## S4 method for signature 'SequenceData'
bamfiles(x)

## S4 method for signature 'SequenceData'
conditions(object)

## S4 method for signature 'SequenceData'
ranges(x)

## S4 method for signature 'SequenceData'
replicates(x)

## S4 method for signature 'SequenceData'
seqinfo(x)

## S4 method for signature 'SequenceData'
sequences(x)
```



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

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

## S4 method for signature 'SequenceData'
dataType(x)

## S4 method for signature 'SequenceDataSet'
show(object)

## S4 method for signature 'SequenceDataSet'
bamfiles(x)

## S4 method for signature 'SequenceDataSet'
conditions(object)

## S4 method for signature 'SequenceDataSet'
names(x)

## S4 method for signature 'SequenceDataSet'
ranges(x)

## S4 method for signature 'SequenceDataSet'
replicates(x)

## S4 method for signature 'SequenceDataSet'
seqinfo(x)

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

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

## S4 method for signature 'SequenceDataSet'
sequences(x)

## S4 method for signature 'SequenceDataList'
show(object)

## S4 method for signature 'SequenceDataList'
bamfiles(x)

## S4 method for signature 'SequenceDataList'
conditions(object)

## S4 method for signature 'SequenceDataList'
names(x)
```

```
## S4 method for signature 'SequenceDataList'
ranges(x)

## S4 method for signature 'SequenceDataList'
replicates(x)

## S4 method for signature 'SequenceDataList'
seqinfo(x)

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

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

## S4 method for signature 'SequenceDataList'
sequences(x)
```

Arguments

<code>x, object</code>	a <code>SequenceData</code> , <code>SequenceDataSet</code> , <code>SequenceDataList</code> or a <code>SequenceDataFrame</code> object.
<code>value</code>	a new <code>seqtype</code> , either "RNA" or "DNA"
<code>bamfiles</code>	a <code>BamFileList</code> .
<code>grl</code>	a <code>GRangesList</code> from <code>exonsBy(..., by = "tx")</code>
<code>sequences</code>	a <code>XStringSet</code> of type <code>RNAStringSet</code> , <code>ModRNAStringSet</code> , <code>DNAStringSet</code> or <code>ModDNAStringSet</code>
<code>param</code>	a ScanBamParam object
<code>args</code>	a list of addition arguments

Value

- `seqinfo`: a `Seqinfo` object (`()`).
- `sequences`: a `RNAstingSet` object or a `RNAString` object for a `SequenceDataFrame`.
- `ranges`: a `GRangesList` object with each element per transcript or a `GRanges` object for a `SequenceDataFrame`.
- `bamfiles`: a `BamFileList` object or a `SimpleList` of `BamFileList` objects for a `SequenceDataList`.

Examples

```
data(e5sd, package="RNAmodR")
# general accessors
seqinfo(e5sd)
sequences(e5sd)
ranges(e5sd)
bamfiles(e5sd)
```

SequenceDataFrame-class

The SequenceDataFrame class

Description

The SequenceDataFrame class is a virtual class and contains data for positions along a single transcript. In addition to being used for returning elements from a SequenceData object, the SequenceDataFrame class is used to store the unlisted data within a [SequenceData](#) object. Therefore, a matching SequenceData and SequenceDataFrame class must be implemented.

The SequenceDataFrame class is derived from the [DataFrame](#) class. To follow the functionality in the S4Vectors package, SequenceDataFrame implements the concept, whereas SequenceDFrame is the implementation for in-memory data representation from which some specific *SequenceDataFrame class derive from, e.g. [CoverageSequenceData](#).

Subsetting of a SequenceDataFrame returns a SequenceDataFrame or DataFrame, if it is subset by a column or row, respectively. The drop argument is ignored for column subsetting.

Usage

```
## S4 method for signature 'SequenceDataFrame'
cbind(..., deparse.level = 1)

## S4 method for signature 'SequenceDataFrame,ANY,ANY,ANY'
x[i, j, ..., drop = TRUE]
```

Arguments

x, i, j, ..., drop, deparse.level
arguments used for [subsetting](#) or [base::cbind](#).

Value

A SequenceDataFrame object or if subset to row a DataFrame

Slots

ranges a [GRanges](#) object each element describing a transcript including its element. The GRanges is constructed from the unlisted results of the [exonsBy\(x, by="tx"\)](#) function. If during construction a GRangesList is provided instead of a character value pointing to a gff3 file or a TxDb object, it must have a comparable structure.

sequence a [XString](#) of type sequencesType from the parent [SequenceData](#) object.

condition conditions along the [BamFileList](#): Either control or treated

replicate replicate number along the BamFileList for each of the condition types.

bamfiles the input bam files as [BamFileList](#)

seqinfo a [Seqinfo](#) describing the available/used chromosomes.

See Also

for an example see [ProtectedEndSequenceData](#) and for more information see [SequenceData](#)

Examples

```
data(e5sd,package="RNAmodR")
# A SequenceDataFrame can is usually constructed by subsetting from
# a SequenceData object
sdf <- e5sd[[1]]
# Its also used to store the unlisted data in a SequenceData object
sdf <- unlist(e5sd) # should probably only used internally
e5sd <- relist(sdf,e5sd)
```

SequenceDataList-class*The SequenceDataList class*

Description

The SequenceDataList class is used to hold SequenceData or SequenceDataSet objects as its elements. It is derived from the [List](#) class.

The SequenceDataList is used to hold data from different sets of aligned reads. This allows multiple methods to be aggregated into one modification detection strategy. Annotation and sequence data must be the same for all elements, however the bam files can be different.

Usage

```
SequenceDataList(...)
```

Arguments

... The elements to be included in the SequenceDataList.

Value

a SequenceDataList

Examples

```
data(psd,package="RNAmodR")
data(e5sd,package="RNAmodR")
sdl <- SequenceDataList(SequenceDataSet(psd,e5sd),e5sd)
```

SequenceDataSet-class *The SequenceDataSet class*

Description

The SequenceDataSet class is used to hold SequenceData objects as its elements. It is derived from the [List](#) class.

The SequenceDataSet is used to hold different data types from the of same aligned reads. The same dataset can be used to generate multiple sets of data types. Bam files, annotation and sequence data must be the same for all elements.

Usage

```
SequenceDataSet(...)
```

Arguments

... The elements to be included in the SequenceDataSet.

Value

a SequenceDataSet

Examples

```
data(psd, package="RNAmodR")
data(e5sd, package="RNAmodR")
sdl <- SequenceDataSet(psd, e5sd)
```

SequenceModDNAStringSetTrack-class
ModDNASequenceTrack

Description

A Gviz compatible [SequenceTrack](#) for showing modified DNA sequences.

Usage

```
ModDNASequenceTrack(sequence, chromosome, genome, name = "SequenceTrack", ...)
```

```
## S4 method for signature 'SequenceModDNAStringSetTrack'
seqnames(x)
```

```
## S4 method for signature 'SequenceModDNAStringSetTrack'
seqlevels(x)
```

Arguments

sequence A character vector or ModDNAString object of length one. The sequence to display. See [SequenceTrack](#).

chromosome, genome, name, ...
 See [SequenceTrack](#).

x A SequenceModDNAStringSetTrack object.

Value

a SequenceModDNAStringSetTrack object

Slots

sequence A ModDNAStringSet object

Examples

```
seq <- ModDNAStringSet(c(chr1 = paste0(alphabet(ModDNAString()),
                                         collapse = "")))
st <- ModDNASequenceTrack(seq)
Gviz::plotTracks(st, chromosome = "chr1", from = 1L, to = 20L)
```

SequenceModRNAStringSetTrack-class
ModRNASequenceTrack

Description

A Gviz compatible [SequenceTrack](#) for showing modified RNA sequences.

Usage

```
ModRNASequenceTrack(sequence, chromosome, genome, name = "SequenceTrack", ...)
```

```
## S4 method for signature 'SequenceModRNAStringSetTrack'
seqnames(x)
```

```
## S4 method for signature 'SequenceModRNAStringSetTrack'
seqlevels(x)
```

Arguments

sequence	A character vector or ModRNAString object of length one. The sequence to display. See SequenceTrack .
chromosome, genome, name, ...	See SequenceTrack .
x	A SequenceModRNAStringSetTrack object.

Value

a SequenceModRNAStringSetTrack object

Slots

sequence A ModRNAStringSet object

Examples

```
seq <- ModRNAStringSet(c(chr1 = paste0(alphabet(ModRNAString()),
                                         collapse = "")))
st <- ModRNASequenceTrack(seq)
Gviz::plotTracks(st, chromosome = "chr1", from = 1L, to = 20L)
```

Description

Depending on data preparation, quality and desired stringency of a modification strategy, settings for cut off parameters or other variables may need to be adjusted. This should be rarely the case, but a function for changing these settings, is implemented as the... settings function.

For changing values the input can be either a list or something coercible to a list. Upon changing a setting, the validity of the value in terms of type(!) and dimensions will be checked.

If settings have been modified after the data was loaded, the data is potentially invalid. To update the data, run the aggregate or the modify function.

Usage

```
settings(x, name = NULL)

settings(x, name) <- value

## S4 method for signature 'Modifier'
settings(x, name = NULL)

## S4 replacement method for signature 'Modifier'
settings(x) <- value

## S4 method for signature 'ModifierSet'
settings(x, name = NULL)

## S4 replacement method for signature 'ModifierSet'
settings(x) <- value
```

Arguments

x	a Modifier or ModifierSet class
name	name of the setting to be returned or set
value	value of the setting to be set

Value

If name is omitted, settings returns a list of all settings. If name is set, settings returns a single settings or NULL, if a value for name is not available.

Examples

```
data(msi, package="RNAmodR")
mi <- msi[[1]]
# returns a list of all settings
settings(mi)
# accesses a specific setting
settings(mi, "minCoverage")
# modification of setting
settings(mi) <- list(minCoverage = 11L)
```

stats

Retrieving information about used reads in RNAmodR

Description

stats returns information about reads used in the RNAmodR analysis. Three modes are available depending on which type of object is provided. If a [SequenceData](#) object is provided, a [BamFile](#) or [BamFileList](#) must be provided as well. If a [Modifier](#) object is used, the bam files returned from the bamfiles function are used. This is also the case, if a [ModifierSet](#) object is used.

Usage

```
stats(x, file, ...)

## S4 method for signature 'SequenceData,BamFile'
stats(x, file, ...)

## S4 method for signature 'SequenceData,BamFileList'
stats(x, file, ...)

## S4 method for signature 'Modifier,missing'
stats(x)

## S4 method for signature 'ModifierSet,missing'
stats(x)
```

Arguments

x a [SequenceData](#), [Modifier](#) or [ModifierSet](#) object
file a [BamFile](#) or [BamFileList](#), if x is a [SequenceData](#) object.
... optional parameters used as stated [here](#) (except minQuality), if x is a [SequenceData](#) object.

Value

a [DataFrame](#), [DataFrameList](#) or [SimpleList](#) with the results in aggregated form

Examples

```
library(RNAmodR.Data)
library(rtracklayer)
sequences <- RNAmodR.Data.example.AAS.fasta()
annotation <- GFF3File(RNAmodR.Data.example.AAS.gff3())
files <- list("SampleSet1" = c(treated = RNAmodR.Data.example.wt.1(),
                             treated = RNAmodR.Data.example.wt.2(),
                             treated = RNAmodR.Data.example.wt.3()),
             "SampleSet2" = c(treated = RNAmodR.Data.example.bud23.1(),
                             treated = RNAmodR.Data.example.bud23.2()),
             "SampleSet3" = c(treated = RNAmodR.Data.example.trm8.1(),
                             treated = RNAmodR.Data.example.trm8.2()))
msi <- ModSetInosine(files, annotation = annotation, sequences = sequences)
# smallest chunk of information
```



```
stats(sequenceData(msi[[1L]]),bamfiles(msi[[1L]])[[1L]])
# partial information
stats(sequenceData(msi[[1L]]),bamfiles(msi[[1L]]))
# the whole stats
stats(msi)
```

subsetByCoord	<i>Subsetting data from a SequenceData, SequenceDataSet, SequenceDataList, Modifier or ModifierSet object.</i>
---------------	--

Description

With the subsetByCoord function data from a SequenceData, SequenceDataSet, SequenceDataList, Modifier or ModifierSet object can be subset to positions as defined in coord.

If coord contains a column mod and x is a Modifier object, it will be filtered to identifiers matching the `modType` of x. To disable this behaviour remove the column mod from coord or set type = NA

labelByCoord functions similarly. It will return a SplitDataFrameList, which matches the dimensions of the aggregated data plus the labels column, which contains logical values to indicate selected positions.

Usage

```
subsetByCoord(x, coord, ...)

labelByCoord(x, coord, ...)

## S4 method for signature 'Modifier,GRanges'
subsetByCoord(x, coord, ...)

## S4 method for signature 'Modifier,GRangesList'
subsetByCoord(x, coord, ...)

## S4 method for signature 'ModifierSet'
subset(x, name, pos = 1L, ...)

## S4 method for signature 'ModifierSet,GRanges'
subsetByCoord(x, coord, ...)

## S4 method for signature 'ModifierSet,GRangesList'
subsetByCoord(x, coord, ...)

## S4 method for signature 'Modifier,GRanges'
labelByCoord(x, coord, ...)

## S4 method for signature 'Modifier,GRangesList'
labelByCoord(x, coord, ...)

## S4 method for signature 'ModifierSet,GRanges'
labelByCoord(x, coord, ...)

## S4 method for signature 'ModifierSet,GRangesList'
```

```
labelByCoord(x, coord, ...)

## S4 method for signature 'SplitDataFrameList,GRanges'
subsetByCoord(x, coord, ...)

## S4 method for signature 'SequenceData'
subset(x, name, pos = 1L, ...)

## S4 method for signature 'SequenceData,GRanges'
subsetByCoord(x, coord, ...)

## S4 method for signature 'SequenceData,GRangesList'
subsetByCoord(x, coord, ...)

## S4 method for signature 'SequenceDataSet'
subset(x, name, pos = 1L, ...)

## S4 method for signature 'SequenceDataSet,GRanges'
subsetByCoord(x, coord, ...)

## S4 method for signature 'SequenceDataSet,GRangesList'
subsetByCoord(x, coord, ...)

## S4 method for signature 'SequenceDataList'
subset(x, name, pos = 1L, ...)

## S4 method for signature 'SequenceDataList,GRanges'
subsetByCoord(x, coord, ...)

## S4 method for signature 'SequenceDataList,GRangesList'
subsetByCoord(x, coord, ...)

## S4 method for signature 'SequenceData,GRanges'
labelByCoord(x, coord, ...)

## S4 method for signature 'SequenceData,GRangesList'
labelByCoord(x, coord, ...)

## S4 method for signature 'SequenceDataSet,GRanges'
labelByCoord(x, coord, ...)

## S4 method for signature 'SequenceDataSet,GRangesList'
labelByCoord(x, coord, ...)

## S4 method for signature 'SequenceDataList,GRanges'
labelByCoord(x, coord, ...)

## S4 method for signature 'SequenceDataList,GRangesList'
labelByCoord(x, coord, ...)
```

Arguments

x	a <code>SequenceData</code> , <code>SequenceDataSet</code> , <code>SequenceDataList</code> , <code>Modifier</code> or <code>ModifierSet</code> object.
coord	coordinates of position to subset to. Either a <code>GRanges</code> or a <code>GRangesList</code> object. For both types the 'Parent' column is expected to match the transcript name.
...	Optional parameters: • <code>type</code>: the modification type used for subsetting. By default this is derived from the <code>modType(x)</code>, but it can be overwritten using <code>type</code>. It must be a valid <code>shortName</code> for a modification according to <code>shortName(ModRNAString())</code> or <code>shortName(ModDNAString())</code> (depending on the type of <code>Modifier</code> class) and of course be present in metadata column <code>mod</code> of <code>coord</code>. To disable subsetting based on type, set <code>type = NA</code>. • <code>flanking</code>: a single integer value to select how many flanking position should be included in the subset (default: <code>flanking = 0L</code>). • <code>merge</code>: <code>TRUE</code> or <code>FALSE</code>: Should the overlapping selections be merged? This is particular important, if <code>flanking</code> value $\neq 0L$ are set. (default: <code>merge = TRUE</code>). • <code>perTranscript</code>: <code>TRUE</code> or <code>FALSE</code>: Should the positions labeled per transcript and not per chromosome? (default: <code>perTranscript = FALSE</code>).
name	Optional: Limit results to one specific transcript.
pos	Optional: Limit results to a specific position.

Value

If 'x' is a

- [SequenceData](#) or [Modifier](#): a `SplitDataFrameList` with elements per transcript.
- [SequenceDataSet](#), [SequenceDataList](#) or [ModifierSet](#): a `SimpleList` of `SplitDataFrameList` with elements per transcript.

Examples

```
data(msi, package="RNAmodR")
mod <- modifications(msi)
coord <- unique(unlist(mod))
coord$score <- NULL
coord$sd <- NULL
subsetByCoord(msi, coord)
```

Index

- * **datasets**
 - RNAmodR-datasets, [32](#)
- * **internal**
 - RNAmodR-internals, [36](#)
 - RNAmodR-package, [3](#)
- .dataTracks, ModInosine, GRanges, GRanges, XStringMethod
 - (ModInosine-internals), [22](#)
- .getData (RNAmodR-internals), [36](#)
- [, SequenceDataFrame, ANY, ANY, ANY-method
 - (SequenceDataFrame-class), [43](#)
- aggregate, [3](#), [7](#), [9](#), [22](#), [23](#), [25](#), [31](#)
- aggregate, Modifier-method (aggregate), [3](#)
- aggregate, ModifierSet-method
 - (aggregate), [3](#)
- aggregate, SequenceData-method
 - (aggregate), [3](#)
- aggregate, SequenceDataList-method
 - (aggregate), [3](#)
- aggregate, SequenceDataSet-method
 - (aggregate), [3](#)
- aggregateData, [10](#), [18](#), [33](#)
- aggregateData (aggregate), [3](#)
- aggregateData, CoverageSequenceData-method
 - (CoverageSequenceData-class), [7](#)
- aggregateData, End3SequenceData-method
 - (EndSequenceData-class), [8](#)
- aggregateData, End5SequenceData-method
 - (EndSequenceData-class), [8](#)
- aggregateData, EndSequenceData-method
 - (EndSequenceData-class), [8](#)
- aggregateData, Modifier-method
 - (aggregate), [3](#)
- aggregateData, ModInosine-method
 - (ModInosine-functions), [21](#)
- aggregateData, NormEnd3SequenceData-method
 - (NormEndSequenceData-class), [22](#)
- aggregateData, NormEnd5SequenceData-method
 - (NormEndSequenceData-class), [22](#)
- aggregateData, PileupSequenceData-method
 - (PileupSequenceData-class), [24](#)
- aggregateData, ProtectedEndSequenceData-method
 - (ProtectedEndSequenceData-class), [30](#)
- aggregateData, SequenceData-method
 - (aggregate), [3](#)
- AnnotationTrack, [28](#)
- BamFile, [48](#)
- BamFileList, [43](#), [48](#)
- bamfiles (Modifier-functions), [13](#)
- bamfiles, Modifier-method
 - (Modifier-functions), [13](#)
- bamfiles, ModifierSet-method
 - (Modifier-functions), [13](#)
- bamfiles, SequenceData-method
 - (SequenceData-functions), [39](#)
- bamfiles, SequenceDataFrame-method
 - (SequenceData-functions), [39](#)
- bamfiles, SequenceDataList-method
 - (SequenceData-functions), [39](#)
- bamfiles, SequenceDataSet-method
 - (SequenceData-functions), [39](#)
- base::cbind, [39](#), [43](#)
- cbind, SequenceData-method
 - (SequenceData-class), [37](#)
- cbind, SequenceDataFrame-method
 - (SequenceDataFrame-class), [43](#)
- combineIntoModstrings, [15](#), [28](#)
- compare, [5](#)
- compare, ModifierSet-method (compare), [5](#)
- compareByCoord (compare), [5](#)
- compareByCoord, ModifierSet, GRanges-method
 - (compare), [5](#)
- compareByCoord, ModifierSet, GRangesList-method
 - (compare), [5](#)
- CompressedSplitDataFrameList, [34](#), [37](#)
- conditions, Modifier-method
 - (Modifier-functions), [13](#)
- conditions, ModifierSet-method
 - (Modifier-functions), [13](#)
- conditions, SequenceData-method
 - (SequenceData-functions), [39](#)
- conditions, SequenceDataFrame-method
 - (SequenceData-functions), [39](#)
- conditions, SequenceDataList-method
 - (SequenceData-functions), [39](#)

- conditions, SequenceDataSet-method
(SequenceData-functions), 39
- constructModRanges
(RNAmodR-development), 33
- constructModRanges, GRanges, DataFrame-method
(RNAmodR-development), 33
- CoverageSequenceData, 37, 43
- CoverageSequenceData
(CoverageSequenceData-class), 7
- CoverageSequenceData-class, 7
- CoverageSequenceDataFrame
(CoverageSequenceData-class), 7
- CoverageSequenceDataFrame-class
(CoverageSequenceData-class), 7
- csd (RNAmodR-datasets), 32
- DataFrame, 6, 37, 43
- DataTrack, 22, 25
- dataType (Modifier-functions), 13
- dataType, Modifier-method
(Modifier-functions), 13
- dataType, ModifierSet-method
(Modifier-functions), 13
- dataType, SequenceData-method
(SequenceData-functions), 39
- dataType, SequenceDataFrame-method
(SequenceData-functions), 39
- DNAModifier-class (Modifier-class), 10
- e3sd (RNAmodR-datasets), 32
- e5sd (RNAmodR-datasets), 32
- End3SequenceData, 37
- End3SequenceData
(EndSequenceData-class), 8
- End3SequenceData-class
(EndSequenceData-class), 8
- End3SequenceDataFrame
(EndSequenceData-class), 8
- End3SequenceDataFrame-class
(EndSequenceData-class), 8
- End5SequenceData, 37
- End5SequenceData
(EndSequenceData-class), 8
- End5SequenceData-class
(EndSequenceData-class), 8
- End5SequenceDataFrame
(EndSequenceData-class), 8
- End5SequenceDataFrame-class
(EndSequenceData-class), 8
- EndSequenceData, 37
- EndSequenceData
(EndSequenceData-class), 8
- EndSequenceData-class, 8
- EndSequenceDataFrame
(EndSequenceData-class), 8
- EndSequenceDataFrame-class
(EndSequenceData-class), 8
- esd (RNAmodR-datasets), 32
- findMod, 10, 18, 33
- findMod (modify), 18
- findMod, Modifier-method (modify), 18
- findMod, ModInosine-method
(ModInosine-functions), 21
- Functions, 19
- functions, 37
- getAggregateData (aggregate), 3
- getAggregateData, Modifier-method
(aggregate), 3
- getData, 18
- getData (RNAmodR-development), 33
- getData, CoverageSequenceData, BamFileList, GRangesList, XStringSet
(CoverageSequenceData-class), 7
- getData, End3SequenceData, BamFileList, GRangesList, XStringSet
(EndSequenceData-class), 8
- getData, End5SequenceData, BamFileList, GRangesList, XStringSet
(EndSequenceData-class), 8
- getData, EndSequenceData, BamFileList, GRangesList, XStringSet
(EndSequenceData-class), 8
- getData, NormEnd3SequenceData, BamFileList, GRangesList, XStringSet
(NormEndSequenceData-class), 22
- getData, NormEnd5SequenceData, BamFileList, GRangesList, XStringSet
(NormEndSequenceData-class), 22
- getData, PileupSequenceData, BamFileList, GRangesList, XStringSet
(PileupSequenceData-class), 24
- getData, ProtectedEndSequenceData, BamFileList, GRangesList, XStringSet
(ProtectedEndSequenceData-class), 30
- getData, SequenceData, BamFileList, GRangesList, XStringSet
(SequenceData-functions), 39
- getDataTrack, 9, 24, 25, 31
- getDataTrack (plotData), 25
- getDataTrack, CoverageSequenceData-method
(CoverageSequenceData-class), 7
- getDataTrack, End3SequenceData-method
(EndSequenceData-class), 8
- getDataTrack, End5SequenceData-method
(EndSequenceData-class), 8
- getDataTrack, EndSequenceData-method
(EndSequenceData-class), 8
- getDataTrack, Modifier-method
(plotData), 25
- getDataTrack, ModInosine-method
(ModInosine-functions), 21

- getDataTrack, NormEnd3SequenceData-method
(NormEndSequenceData-class), 22
- getDataTrack, NormEnd5SequenceData-method
(NormEndSequenceData-class), 22
- getDataTrack, PileupSequenceData-method
(PileupSequenceData-class), 24
- getDataTrack, ProtectedEndSequenceData-method
(ProtectedEndSequenceData-class), 30
- getDataTrack, SequenceData-method
(plotData), 25
- getDataTrack, SequenceDataList-method
(plotData), 25
- getDataTrack, SequenceDataSet-method
(plotData), 25
- getListElement, SequenceDataList-method
(RNAmoR-internals), 36
- getListElement, SequenceDataSet-method
(RNAmoR-internals), 36
- GRanges, 5, 43
- GRangesList, 5
- hasAggregateData (aggregate), 3
- hasAggregateData, Modifier-method
(aggregate), 3
- here, 48
- ICE-Seq (ModInosine), 19
- Inosine (ModInosine), 19
- IntegerList, 34
- labelByCoord (subsetByCoord), 49
- labelByCoord, Modifier, GRanges-method
(subsetByCoord), 49
- labelByCoord, Modifier, GRangesList-method
(subsetByCoord), 49
- labelByCoord, ModifierSet, GRanges-method
(subsetByCoord), 49
- labelByCoord, ModifierSet, GRangesList-method
(subsetByCoord), 49
- labelByCoord, SequenceData, GRanges-method
(subsetByCoord), 49
- labelByCoord, SequenceData, GRangesList-method
(subsetByCoord), 49
- labelByCoord, SequenceDataList, GRanges-method
(subsetByCoord), 49
- labelByCoord, SequenceDataList, GRangesList-method
(subsetByCoord), 49
- labelByCoord, SequenceDataSet, GRanges-method
(subsetByCoord), 49
- labelByCoord, SequenceDataSet, GRangesList-method
(subsetByCoord), 49
- labelByCoord, SplitDataFrameList, GRanges-method
(RNAmoR-internals), 36
- List, 44
- mainScore (Modifier-functions), 13
- mainScore, Modifier-method
(Modifier-functions), 13
- mainScore, ModifierSet-method
(Modifier-functions), 13
- ModDNASequenceTrack
(SequenceModDNAStringSetTrack-class), 45
- ModDNAString, 12
- ModifierSet, 4
- modifications (modify), 18
- modifications, Modifier-method (modify), 18
- modifications, ModifierSet-method
(Modifier-functions), 13
- Modifier, 3–5, 15, 16, 18, 19, 21, 34, 48, 51
- Modifier (Modifier-class), 10
- Modifier, BamFileList-method
(Modifier-class), 10
- Modifier, character-method
(Modifier-class), 10
- Modifier, list-method (Modifier-class), 10
- Modifier, SequenceData-method
(Modifier-class), 10
- Modifier, SequenceDataList-method
(Modifier-class), 10
- Modifier, SequenceDataSet-method
(Modifier-class), 10
- Modifier-class, 10
- Modifier-functions, 13
- ModifierInosine (ModInosine), 19
- ModifierSet, 5, 19, 21, 48, 51
- ModifierSet (ModifierSet-class), 16
- ModifierSet, BamFileList-method
(ModifierSet-class), 16
- ModifierSet, character-method
(ModifierSet-class), 16
- ModifierSet, list-method
(ModifierSet-class), 16
- ModifierSet, Modifier-method
(ModifierSet-class), 16
- ModifierSet-class, 16
- modifierType (Modifier-functions), 13
- modifierType, Modifier-method
(Modifier-functions), 13
- modifierType, ModifierSet-method
(Modifier-functions), 13
- modify, 18, 22

- modify, Modifier-method (modify), 18
- modify, ModifierSet-method (modify), 18
- ModInosine, 19
- ModInosine-class (ModInosine), 19
- ModInosine-functions, 21
- ModInosine-internals, 22
- ModRNASequenceTrack
 - (SequenceModRNAStringSetTrack-class), 46
- ModRNAString, 12
- ModSetInosine (ModInosine), 19
- ModSetInosine-class (ModInosine), 19
- modType, 49
- modType (Modifier-functions), 13
- modType, Modifier-method
 - (Modifier-functions), 13
- modType, ModifierSet-method
 - (Modifier-functions), 13
- msi (RNAmodR-datasets), 32
- names, Modifier-method
 - (Modifier-functions), 13
- names, SequenceDataList-method
 - (SequenceData-functions), 39
- names, SequenceDataSet-method
 - (SequenceData-functions), 39
- ne3sd (RNAmodR-datasets), 32
- ne5sd (RNAmodR-datasets), 32
- NormEnd3SequenceData
 - (NormEndSequenceData-class), 22
- NormEnd3SequenceData-class
 - (NormEndSequenceData-class), 22
- NormEnd3SequenceDataFrame
 - (NormEndSequenceData-class), 22
- NormEnd3SequenceDataFrame-class
 - (NormEndSequenceData-class), 22
- NormEnd5SequenceData, 37
- NormEnd5SequenceData
 - (NormEndSequenceData-class), 22
- NormEnd5SequenceData-class
 - (NormEndSequenceData-class), 22
- NormEnd5SequenceDataFrame
 - (NormEndSequenceData-class), 22
- NormEnd5SequenceDataFrame-class
 - (NormEndSequenceData-class), 22
- NormEndSequenceData-class, 22
- NumericList, 34
- parallel_slot_names, SequenceDataList-method
 - (RNAmodR-internals), 36
- parallel_slot_names, SequenceDataSet-method
 - (RNAmodR-internals), 36
- performance, 29
- pesd (RNAmodR-datasets), 32
- PileupSequenceData, 37
- PileupSequenceData
 - (PileupSequenceData-class), 24
- PileupSequenceData-class, 24
- PileupSequenceDataFrame
 - (PileupSequenceData-class), 24
- PileupSequenceDataFrame-class
 - (PileupSequenceData-class), 24
- pileupToCoverage
 - (PileupSequenceData-class), 24
- pileupToCoverage, PileupSequenceData-method
 - (PileupSequenceData-class), 24
- plotCompare (compare), 5
- plotCompare, ModifierSet-method
 - (compare), 5
- plotCompareByCoord (compare), 5
- plotCompareByCoord, ModifierSet, GRanges-method
 - (compare), 5
- plotCompareByCoord, ModifierSet, GRangesList-method
 - (compare), 5
- plotData, 21, 25, 33
- plotData, Modifier-method (plotData), 25
- plotData, ModifierSet-method (plotData), 25
- plotData, ModInosine-method
 - (ModInosine-functions), 21
- plotData, ModSetInosine-method
 - (ModInosine-functions), 21
- plotData, SequenceData-method
 - (plotData), 25
- plotData, SequenceDataList-method
 - (plotData), 25
- plotData, SequenceDataSet-method
 - (plotData), 25
- plotDataByCoord, 22
- plotDataByCoord (plotData), 25
- plotDataByCoord, Modifier, GRanges-method
 - (plotData), 25
- plotDataByCoord, ModifierSet, GRanges-method
 - (plotData), 25
- plotDataByCoord, ModInosine, GRanges-method
 - (ModInosine-functions), 21
- plotDataByCoord, ModSetInosine, GRanges-method
 - (ModInosine-functions), 21
- plotDataByCoord, SequenceData, GRanges-method
 - (plotData), 25
- plotDataByCoord, SequenceDataList, GRanges-method
 - (plotData), 25
- plotDataByCoord, SequenceDataSet, GRanges-method
 - (plotData), 25
- plotROC, 28

- plotROC, Modifier-method (plotROC), 28
- plotROC, ModifierSet-method (plotROC), 28
- prediction, 29
- ProtectedEndSequenceData, 37, 43
- ProtectedEndSequenceData
 - (ProtectedEndSequenceData-class), 30
- ProtectedEndSequenceData-class, 30
- ProtectedEndSequenceDataFrame
 - (ProtectedEndSequenceData-class), 30
- ProtectedEndSequenceDataFrame-class
 - (ProtectedEndSequenceData-class), 30
- psd (RNAmodR-datasets), 32
- ranges, Modifier-method
 - (Modifier-functions), 13
- ranges, ModifierSet-method
 - (Modifier-functions), 13
- ranges, SequenceData-method
 - (SequenceData-functions), 39
- ranges, SequenceDataFrame-method
 - (SequenceData-functions), 39
- ranges, SequenceDataList-method
 - (SequenceData-functions), 39
- ranges, SequenceDataSet-method
 - (SequenceData-functions), 39
- rbind, SequenceData-method
 - (SequenceData-class), 37
- replicates (SequenceData-functions), 39
- replicates, Modifier-method
 - (Modifier-functions), 13
- replicates, ModifierSet-method
 - (Modifier-functions), 13
- replicates, SequenceData-method
 - (SequenceData-functions), 39
- replicates, SequenceDataFrame-method
 - (SequenceData-functions), 39
- replicates, SequenceDataList-method
 - (SequenceData-functions), 39
- replicates, SequenceDataSet-method
 - (SequenceData-functions), 39
- RNAModifier-class (Modifier-class), 10
- RNAmodR, 31
- RNAmodR-datasets, 32
- RNAmodR-development, 33
- RNAmodR-internals, 36
- RNAmodR-package, 3
- ScanBamParam, 42
- sd1 (RNAmodR-datasets), 32
- sds (RNAmodR-datasets), 32
- Seqinfo, 12, 18, 20, 39, 43
- seqinfo, Modifier-method
 - (Modifier-functions), 13
- seqinfo, ModifierSet-method
 - (Modifier-functions), 13
- seqinfo, SequenceData-method
 - (SequenceData-functions), 39
- seqinfo, SequenceDataFrame-method
 - (SequenceData-functions), 39
- seqinfo, SequenceDataList-method
 - (SequenceData-functions), 39
- seqinfo, SequenceDataSet-method
 - (SequenceData-functions), 39
- seqlevels, SequenceModDNAStringSetTrack-method
 - (SequenceModDNAStringSetTrack-class), 45
- seqlevels, SequenceModRNAStringSetTrack-method
 - (SequenceModRNAStringSetTrack-class), 46
- seqnames, SequenceModDNAStringSetTrack-method
 - (SequenceModDNAStringSetTrack-class), 45
- seqnames, SequenceModRNAStringSetTrack-method
 - (SequenceModRNAStringSetTrack-class), 46
- seqtype, Modifier-method
 - (Modifier-functions), 13
- seqtype, ModifierSet-method
 - (Modifier-functions), 13
- seqtype, SequenceData-method
 - (SequenceData-functions), 39
- seqtype, SequenceDataFrame-method
 - (SequenceData-functions), 39
- seqtype, SequenceDataList-method
 - (SequenceData-functions), 39
- seqtype, SequenceDataSet-method
 - (SequenceData-functions), 39
- seqtype<-, SequenceData-method
 - (SequenceData-functions), 39
- seqtype<-, SequenceDataFrame-method
 - (SequenceData-functions), 39
- seqtype<-, SequenceDataList-method
 - (SequenceData-functions), 39
- seqtype<-, SequenceDataSet-method
 - (SequenceData-functions), 39
- SequenceData, 3–5, 7, 9, 18, 23, 25, 30, 31, 33, 37, 43, 48, 51
- SequenceData (SequenceData-class), 37
- sequenceData (Modifier-functions), 13
- SequenceData, character, BSgenome-method
 - (SequenceData-class), 37
- SequenceData, character, character-method

- (SequenceData-class), 37
- SequenceData, character, FaFile-method
(SequenceData-class), 37
- SequenceData, GFF3File, BSgenome-method
(SequenceData-class), 37
- SequenceData, GFF3File, character-method
(SequenceData-class), 37
- SequenceData, GFF3File, FaFile-method
(SequenceData-class), 37
- SequenceData, GRangesList, BSgenome-method
(SequenceData-class), 37
- SequenceData, GRangesList, character-method
(SequenceData-class), 37
- SequenceData, GRangesList, FaFile-method
(SequenceData-class), 37
- sequenceData, Modifier-method
(Modifier-functions), 13
- SequenceData, TxDb, BSgenome-method
(SequenceData-class), 37
- SequenceData, TxDb, character-method
(SequenceData-class), 37
- SequenceData, TxDb, FaFile-method
(SequenceData-class), 37
- SequenceData-class, 37
- SequenceData-functions, 39
- SequenceDataFrame, 7, 9, 23, 25, 31
- SequenceDataFrame
(SequenceDataFrame-class), 43
- SequenceDataFrame-class, 43
- SequenceDataList, 5, 51
- SequenceDataList
(SequenceDataList-class), 44
- SequenceDataList-class, 44
- SequenceDataSet, 5, 51
- SequenceDataSet
(SequenceDataSet-class), 44
- SequenceDataSet-class, 44
- SequenceDFrame-class
(SequenceDataFrame-class), 43
- SequenceModDNASTringSetTrack
(SequenceModDNASTringSetTrack-class), 45
- SequenceModDNASTringSetTrack-class, 45
- SequenceModRNASTringSetTrack
(SequenceModRNASTringSetTrack-class), 46
- SequenceModRNASTringSetTrack-class, 46
- sequences (Modifier-functions), 13
- sequences, Modifier-method
(Modifier-functions), 13
- sequences, ModifierSet-method
(Modifier-functions), 13
- sequences, SequenceData-method
(SequenceData-functions), 39
- sequences, SequenceDataFrame-method
(SequenceData-functions), 39
- sequences, SequenceDataList-method
(SequenceData-functions), 39
- sequences, SequenceDataSet-method
(SequenceData-functions), 39
- SequenceTrack, 28, 45, 46
- settings, 12, 13, 16, 21, 22, 47
- settings, Modifier-method (settings), 47
- settings, ModifierSet-method (settings), 47
- settings<- (settings), 47
- settings<-, Modifier-method (settings), 47
- settings<-, ModifierSet-method (settings), 47
- settings<-, ModInosine-method
(ModInosine-functions), 21
- show, Modifier-method
(Modifier-functions), 13
- show, ModifierSet-method
(Modifier-functions), 13
- show, SequenceData-method
(SequenceData-functions), 39
- show, SequenceDataFrame-method
(SequenceData-functions), 39
- show, SequenceDataList-method
(SequenceData-functions), 39
- show, SequenceDataSet-method
(SequenceData-functions), 39
- stats, 48
- stats, Modifier, missing-method (stats), 48
- stats, ModifierSet, missing-method (stats), 48
- stats, SequenceData, BamFile-method (stats), 48
- stats, SequenceData, BamFileList-method (stats), 48
- subset, Modifier-method
(RNAModR-internals), 36
- subset, ModifierSet-method
(subsetByCoord), 49
- subset, SequenceData-method
(subsetByCoord), 49
- subset, SequenceDataList-method
(subsetByCoord), 49
- subset, SequenceDataSet-method
(subsetByCoord), 49
- subsetByCoord, 6, 49

subsetByCoord, Modifier, GRanges-method
 (subsetByCoord), [49](#)
subsetByCoord, Modifier, GRangesList-method
 (subsetByCoord), [49](#)
subsetByCoord, ModifierSet, GRanges-method
 (subsetByCoord), [49](#)
subsetByCoord, ModifierSet, GRangesList-method
 (subsetByCoord), [49](#)
subsetByCoord, SequenceData, GRanges-method
 (subsetByCoord), [49](#)
subsetByCoord, SequenceData, GRangesList-method
 (subsetByCoord), [49](#)
subsetByCoord, SequenceDataList, GRanges-method
 (subsetByCoord), [49](#)
subsetByCoord, SequenceDataList, GRangesList-method
 (subsetByCoord), [49](#)
subsetByCoord, SequenceDataSet, GRanges-method
 (subsetByCoord), [49](#)
subsetByCoord, SequenceDataSet, GRangesList-method
 (subsetByCoord), [49](#)
subsetByCoord, SplitDataFrameList, GRanges-method
 (subsetByCoord), [49](#)
subsetting, [43](#)

validAggregate (Modifier-functions), [13](#)
validAggregate, Modifier-method
 (Modifier-functions), [13](#)
validModification (Modifier-functions),
 [13](#)
validModification, Modifier-method
 (Modifier-functions), [13](#)

XString, [43](#)