Package: stringtools (via r-universe)

May 11, 2026

Type Package

Title Tools for working with strings in R

Version 0.02

Date 2020-05-03

Author Sebastian Kranz

Maintainer Sebastian Kranz <sebastian.kranz@uni-ulm.de>

Description Tools for working with strings in R

License GPL >= 2.0

Depends stringr

Suggests data.table

Config/pak/sysreqs libicu-dev

Repository https://repboxr.r-universe.dev

Date/Publication 2022-05-04 20:28:27 UTC

RemoteUrl https://github.com/skranz/stringtools

RemoteRef master

RemoteSha 8a37e75f057c7af3a0b115bd1e82ae66c91a0579

Contents

adapt.blocks.after.replace *Helper function*

Description

Helper function

Usage

```
adapt.blocks.after.replace(block, ...)
```

adapt.pos.after.replace *Helper function*

Description

Helper function

Usage

```
adapt.pos.after.replace(pos, left, len.old, len.new)
```

blocks.add.level.0 *Add level 0 to blocks*

Description

Add level 0 to blocks

Usage

```
blocks.add.level.0(blocks, str, end = nchar(str))
```

char.matrix.to.str *converts a matrix of of single chars in a vector of one string per row*

Description

converts a matrix of of single chars in a vector of one string per row

Usage

```
char.matrix.to.str(mat, collapse = "")
```

`char.vector.to.str` *converts a vector of chars into a single string or multiple strings, broken by sep*

Description

converts a vector of chars into a single string or multiple strings, broken by sep

Usage

```
char.vector.to.str(vec, sep = NULL, collapse = "")
```

`check.str.par` *Check if parameter to a str function have allowed dimensions*

Description

Check if parameter to a str function have allowed dimensions

Usage

```
check.str.par(str, para)
```

`combine.pos.and` *Combine two positions via logical AND*

Description

Combine two positions via logical AND

Usage

```
combine.pos.and(pos1, pos2, return.ind = FALSE)
```

`combine.pos.list.and` *Combine a list of positions via logical AND*

Description

Combine a list of positions via logical AND

Usage

```
combine.pos.list.and(pos.list, return.ind = TRUE)
```

cumsum.ignore *Cummulative sum of number of characters that are ignored*

Description

Cummulative sum of number of characters that are ignored

Usage

```
## S3 method for class 'ignore'
cumsum(ignore)
```

Examples

```
## Not run:
  str =c("Now that is a nice matrix. Not?","but short!")
  ignore = rep(FALSE,max(nchar(str)))
  ignore[c(4:5,8:20)] = TRUE
  ignore
  cumsum.ignore(ignore)
  ignore = matrix(ignore,nrow=NROW(str),ncol=length(ignore),byrow=TRUE)
  cumsum.ignore(ignore)

## End(Not run)
```

get.ignore *Gets a logical ignore vector or list from pos matrices*

Description

Gets a logical ignore vector or list from pos matrices

Usage

```
get.ignore(ignore = NULL, ignore.pos = NULL, only.pos = NULL,
  end = NULL, str = NULL)
```

`has.substr` *Returns for every element of str whether there is a match with pattern works similar than grepl*

Description

Returns for every element of str whether there is a match with pattern works similar than grepl

Usage

```
has.substr(str, pattern, fixed = TRUE, perl = FALSE, ignore = NULL)
```

Examples

```
## Not run:
  str = c("12347382709")
  pattern = c("a","4","56","34","766","b")
  has.substr(str,pattern)

## End(Not run)
```

`ignore.and.complement.pos` *Transforms a boolean vector ignore to a pos matrix and its complement*

Description

Transforms a boolean vector ignore to a pos matrix and its complement

Usage

```
ignore.and.complement.pos(ignore = NULL, ignore.pos = NULL,
  only.pos = NULL)
```

Value

A list first element is the pos and complement matrix. The second element is a vector

ignore.to.pos *Transforms a boolean vector ignore to a pos matrix*

Description

Transforms a boolean vector ignore to a pos matrix

Usage

```
ignore.to.pos(ignore, complement = FALSE)
```

Examples

```
## Not run:
  ignore = c(TRUE,FALSE,TRUE,TRUE,FALSE,TRUE,FALSE)
  ignore.to.pos(ignore)

  ignore = list(c(TRUE,FALSE,TRUE,TRUE,FALSE,TRUE,FALSE),c(TRUE),c(),c(FALSE,FALSE))
  ignore.to.pos(ignore)
  ignore = c(TRUE,FALSE,TRUE,TRUE,FALSE,TRUE,FALSE)
  ignore.and.complement.pos(ignore)


## End(Not run)
```

merge.lines *Combines c("A","B") into a single string seperated by line breaks*

Description

Combines c("A","B") into a single string seperated by line breaks

Usage

```
## S3 method for class 'lines'
merge(txt, collapse = "\n")
```

Examples

```
## Not run:
  merge = merge.lines(c("A","B"))
  merge
  sep.lines(merge)



## End(Not run)
```

`pos.complement` *Returns the complement of a pos matrix again as a pos matrix*

Description

Returns the complement of a pos matrix again as a pos matrix

Usage

```
pos.complement(pos, is.sorted = FALSE, start = 1, end = NULL,
  keep.pos = FALSE, str = NULL)
```

Examples

```
## Not run:
  pos = matrix(c(
    2,2,
    5,8,
    10,10
  ),3,2,byrow=TRUE)
  pos.complement(pos)
  pos.complement(pos,keep.pos=TRUE)

## End(Not run)
```

`pos.to.ignore` *Transforms a pos matrix into a logical ignore vector Warning length is not*

Description

Transforms a pos matrix into a logical ignore vector Warning length is not

Usage

```
pos.to.ignore(pos, end = 1000000L, complement = FALSE, str = NULL)
```

`pos.with.complement` *Returns a pos matrix combined with its complement. The matrix has an attribute "complement" which is a logical vector indicating whether a row in the matrix is the original matrix or a complement*

Description

Returns a pos matrix combined with its complement. The matrix has an attribute "complement" which is a logical vector indicating whether a row in the matrix is the original matrix or a complement

Usage

```
pos.with.complement(pos, is.sorted = FALSE, end = NULL)
```

Examples

```
## Not run:
  pos = matrix(c(
    2,2,
    5,8,
    10,10
  ),3,2,byrow=TRUE)
  pos.with.complement(pos)

## End(Not run)
```

`regexp.fixed` *strings will be treated as fixed constant in regex*

Description

strings will be treated as fixed constant in regex

Usage

```
regexp.fixed(str, fixed = TRUE)
```

Arguments

`str`	a string vector
`fixed`	if FALSE just return str

Value

transformed string vector of same size as str

Examples

```
## Not run:
  str = c("A.","*")
  # regexp.fixed transforms strings
  regexp.fixed(str)
  # fixed in stringr flags strings instead
  fixed(str)

## End(Not run)
```

`replace.german.umlaute` *replaces German Umlaute with ascii letters oe, ue, ae*

Description

replaces German Umlaute with ascii letters oe, ue, ae

Usage

```
replace.german.umlaute(txt = readLines(file), file = NULL,
  write.file = !is.null(file))
```

`sep.lines` *transforms a single string with line breaks into a vector with one element for each line*

Description

transforms a single string with line breaks into a vector with one element for each line

Usage

```
sep.lines(txt, collapse = "\n")
```

`str.at.pos` *pos is a matrix or a list of matrices specifying positions as returned by str.locate.all*

Description

pos is a matrix or a list of matrices specifying positions as returned by str.locate.all

Usage

```
## S3 method for class 'at.pos'
str(str, pos)
```

Examples

```
## Not run:
  str = c("012ab0121","abce","0had112bb1")
  pos = str.locate.all(str,"[a-z]*",fixed=FALSE)
  pos
  str.at.pos(str,pos)
  return(ret)

## End(Not run)
```

`str.between` *Returns the between the first occurence of start and the first occurence of end*

Description

Returns the between the first occurence of start and the first occurence of end

Usage

```
## S3 method for class 'between'
str(str, start, end, ...)
```

Examples

```
## Not run:
  str = c("a * (4+3)","b+3)+1","(a*3+(1+2))", ")")
  str.between(str,"(",")")

  str.between("#< type and","#< "," ")


## End(Not run)
```

str.blocks.pos *Returns a pos matrix indicating blocks like brackets ( ) or quoted parts "text"*

Description

We allow for nested blocks. The position matrix also has an attribute level that describes the level of each block

Usage

```
## S3 method for class 'blocks.pos'
str(str, start, end, ignore = NULL,
  ignore.start = ignore, ignore.end = ignore, fixed = TRUE,
  fixed.start = fixed, fixed.end = fixed)
```

Examples

```
## Not run:
  str = '1+(5*(2+3)+(2+(4-1)))'
#        12345678901234567890
  str.blocks.pos(str,"(",")")


## End(Not run)
```

str.detect *Just a synonym for has.substr*

Description

Just a synonym for has.substr

Usage

```
## S3 method for class 'detect'
str(str, pattern, fixed = TRUE, perl = FALSE,
  ignore = NULL)
```

`str.ends.with` *Returns als elements of txt that end with pattern*

Description

Returns als elements of txt that end with pattern

Returns als elements of txt that end with pattern

Usage

```
## S3 method for class 'ends.with'
str(txt, pattern)

## S3 method for class 'ends.with'
str(txt, pattern)
```

Examples

```
## Not run:
  str = c("Hi how are you", "hi", "now what", "Hi")
  str.ends.with(str,"you")

## End(Not run)
```

`str.extract.all` *Returns a list that contains for each element of str (or pattern) a vector of all substrings that match the pattern. If for a string no element is matched an empty list is returned*

Description

Returns a list that contains for each element of str (or pattern) a vector of all substrings that match the pattern. If for a string no element is matched an empty list is returned

Usage

```
## S3 method for class 'extract.all'
str(str, pattern, fixed = FALSE, perl = FALSE,
  ignore = NULL)
```

str.extract.first — *Returns a vector that contains for each element of str (or pattern) the first substring that matches pattern or NA if no match could be found*

Description

Returns a vector that contains for each element of str (or pattern) the first substring that matches pattern or NA if no match could be found

Usage

```
## S3 method for class 'extract.first'
str(str, pattern, fixed = FALSE, perl = FALSE,
  ignore = NULL)
```

str.find — *Find substring positions or matches*

Description

A general wrapper from str.locate.first, str.locate.all, str.extract.first, str.extract.all

Usage

```
## S3 method for class 'find'
str(str, pattern, fixed = TRUE, first = FALSE,
  all = !first, simplify = TRUE, matches = FALSE, ...)
```

Arguments

str	vector of strings that will be searched
pattern	a search pattern
fixed	if FALSE perform regular expression search
first	shall only the first element be returned
all	shall all elements be returned
simplify	try to simplify a list return
matches	if FALSE pos will returned, otherwise the extracted substrings

`str.inpos` *Returns a logical vector with TRUE for every character of str that is in pos*

Description

Returns a logical vector with TRUE for every character of str that is in pos

Usage

```
## S3 method for class 'inpos'
str(str, pos)
```

`str.left` *keeps characters on left*

Description

keeps characters on left

Usage

```
## S3 method for class 'left'
str(str, len = 1)
```

`str.left.of` *Returns the substring left to the first occurence of pattern*

Description

Returns the substring left to the first occurence of pattern

Usage

```
## S3 method for class 'left.of'
str(str, pattern, ..., not.found = str)
```

Examples

```
## Not run:
  str = c("a = 5","b+3","a+3 = 4 = 3", "=")
  str.left.of(str,"=")
  str.left.of(str,"=", not.found=NA)
  str.right.of(str,"=")
  str.right.of(str,"=", not.found=NA)

## End(Not run)
```

str.len *a synonym for nchar*

Description

a synonym for nchar

Usage

```
## S3 method for class 'len'
str(str)
```

str.list.to.regexp.or *Transforms a vector of strings like c("A","B","C") into "A|B|C"*

Description

Transforms a vector of strings like c("A","B","C") into "A|B|C"

Usage

```
## S3 method for class 'list.to.regexp.or'
str(str, fixed = TRUE)
```

Arguments

str	a string vector
fixed	if TRUE treats str as constants in regexp

Value

a single string

Examples

```
## Not run:
  greek=c("alpha","beta","gamma")
  str.list.to.regexp.or(greek)

## End(Not run)
```

`str.locate.all` *Finds start and end positions of all substrings that match pattern*

Description

Finds start and end positions of all substrings that match pattern

Usage

```
## S3 method for class 'locate.all'
str(str, pattern, fixed = TRUE, perl = FALSE,
  ignore = NULL, ignore.pos = NULL, only.pos = NULL)
```

Arguments

`ignore.pos`	a logical vector or logical matrix indicating which locations of str shall be ignored in the search

Value

a list, of matrices n * 2 matrices. The first column is the start position, second column end position of each match

Examples

```
## Not run:
  str.locate.all("0120121","1")
  str.locate.all(c("0120121","abce","011"),"1")

  str = c("0120121","abce","011bb1")
  #str = c("0120121")
  ignore = rep(FALSE,max(nchar(str)))
  ignore[c(2:4)] = TRUE
  str.locate.all(str,"1",ignore=ignore)
  ignore.pos = rbind(c(2,4))
  str.locate.all(str,"1",ignore.pos=ignore.pos)

  str.locate.all(str,c("1","b","a"),ignore=ignore)

  str = c("0120121")
  str.locate.all(str,c("1","b","2"),ignore=ignore)

  # Compare regular expression matching
  str = c("012ab0121","adch3b23","0123")
  gregexpr("[ab]*",str)
  str_locate_all(str,"[ab]*")
  str.locate.first(str,"[ab]*",fixed=FALSE)
  str.locate.all(str,"[ab]*",fixed=FALSE)
  str.locate.all(str,c("[ab]*","3+","0*"),fixed=FALSE)
```

```
  str.locate.first(str,c("[ab]*","2","0*"),fixed=FALSE)
  str.locate.all(str,"ab",fixed=FALSE)

  return(ret)

## End(Not run)
```

`str.locate.at.end` *Locate a pattern at the end of str*

Description

Locate a pattern at the end of str

Usage

```
## S3 method for class 'locate.at.end'
str(str, pattern, fixed = TRUE)
```

Examples

```
## Not run:
  str.locate.at.end(c("0123456012","1230","012","01bsf"),"012")

## End(Not run)
```

`str.locate.at.start` *Locate a pattern at the start of strings*

Description

Locate a pattern at the start of strings

Usage

```
## S3 method for class 'locate.at.start'
str(str, pattern, fixed = TRUE)
```

Examples

```
## Not run:
  str.locate.at.start(c("0123456012","1230","012012","01bsf"),"012")
  str.locate.at.start("0123456",c("012","0","1"))
  str.locate.at.end(c("0123456012","1230","012","01bsf"),"012")


## End(Not run)
```

`str.locate.first` *Finds start and end positions of first substring that matches pattern*

Description

Finds start and end positions of first substring that matches pattern

Usage

```
## S3 method for class 'locate.first'
str(str, pattern, fixed = TRUE, perl = FALSE,
  ignore = NULL, ignore.pos = NULL, only.pos = NULL)
```

Arguments

`ignore.pos`	a logical vector or logical matrix indicating which locations of str shall be ignored in the search

Value

single.return is a 1*2 matrix. First column start position, second column end position

Examples

```
## Not run:


  str.locate.first("Hello","l")
  str.locate.first(c("Hello","What","lol"),"l")
  str.locate.first("Hello",c("l","e"))
  str.locate.first(c("Hello","What","lol"),c("l","g","o"))


  str = "Hello ernie!"
  ignore = rep(FALSE,max(nchar(str)))
  ignore[c(2:4)] = TRUE
  pos = str.locate.first(str,"e",ignore=ignore)
  pos
  str.split.at.pos(str,pos[,1],keep.pos=TRUE)

  ignore.pos = cbind(2,4)
  pos = str.locate.first(str,"e",ignore.pos=ignore.pos)

  pos
  str.split.at.pos(str,pos[,1],keep.pos=TRUE)


  str.detect(str,c("A","[a-z]*"),fixed=FALSE)
```

```
  str = c("Hello ernie","abcdefg","hello erna")
  pos = str.locate.first(str,"e",ignore=ignore)
  pos
  str.split.at.pos(str,pos,keep.pos=TRUE,pos.mat.like.list=TRUE)

  # Compare regular expression matching
  str = c("012ab0121","adch3b23","0123")
  regexpr("[ab]*",str)
  gregexpr("[ab]*",str)
  gregexpr("[ab]*",str,perl=TRUE)
  str_locate(str,c("b"))
  str_locate(str,"[ab]*")
  str_locate_all(str,"[ab]*")


  str.locate.first(str,"[ab]*",fixed=FALSE)
  str.detect(str,"[ab]*",fixed=FALSE)

## End(Not run)
```

---

str.matches.pattern *Check if str completely matches a pattern (not just a substring)*

Description

Check if str completely matches a pattern (not just a substring)

Usage

```
## S3 method for class 'matches.pattern'
str(str, pattern, fixed = TRUE)
```

---

str.number.matches *Returns the number of matches of pattern in each element of str*

Description

Returns the number of matches of pattern in each element of str

Usage

```
## S3 method for class 'number.matches'
str(str, pattern, ...)
```

`str.remove.ends` *remove charcaters on left and right of a string str.remove.ends(c("ABCDEF","01"),1,3) returns c("BC","")*

Description

remove charcaters on left and right of a string str.remove.ends(c("ABCDEF","01"),1,3) returns c("BC","")

Usage

```
## S3 method for class 'remove.ends'
str(str, left = 0, right = 0)
```

Examples

```
## Not run:
  str.remove.ends(c("ABCDEF","01345"),1,3)
  str.remove.ends(c("ABCDEF"),1:2,1:2)
  str.remove.ends(c("ABCDEF","01345"),1:2,1)
  # The following calls throw errors!
  str.remove.ends(c("ABCDEF","01345"),1:2,1:3)
  str.remove.ends(c("ABCDEF","01345","NOW!"),1:2,1)

## End(Not run)
```

`str.remove.ignore` *ignore is a logical vector or matrix stating which char positions shall be ignored the function removes the substrings for which ignore=TRUE*

Description

ignore is a logical vector or matrix stating which char positions shall be ignored the function removes the substrings for which ignore=TRUE

Usage

```
## S3 method for class 'remove.ignore'
str(str, ignore)
```

Examples

```
## Not run:
  str =c("1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ","1234567890")
  ignore = rep(FALSE,max(nchar(str)))
  ignore[c(4:5,8:20)] = TRUE
  str
  str.remove.ignore(str,ignore)

## End(Not run)
```

`str.replace` *Replaces in str every occurence of pattern by replacement*

Description

Replaces in str every occurence of pattern by replacement

Usage

```
## S3 method for class 'replace'
str(str, pattern, replacement, fixed = TRUE, perl = FALSE,
  ignore = NULL, ignore.pos = NULL, only.pos = NULL,
  ignore.pattern = "_IGNORE_", ...)
```

Arguments

`str`	the string to replaced
`pattern`	the substring to be replaced
`replacment`	the new substrings

Value

a string

Examples

```
## Not run:
  str = c("12345678901234567890")
  pattern = c("34","12")
  replacement = c("AB","Holla die Waldfee")
  pos = cbind(1,10)
  str.replace(str,pattern,replacement, ignore.pos=pos)
  str.replace(str,pattern,replacement, only.pos=pos)
  str.replace(str,pattern,replacement)

  str = "int{5*2}*{2*3}"
  pattern = "int{_IGNORE_}"
  replacement = "integer{_IGNORE_}"
```

```
  pos = cbind(c(5,11),c(7,13))
  str.replace(str,pattern,replacement, ignore.pos=pos)

## End(Not run)
```

`str.replace.at.pos` *replace a string at the positions specified by pos*

Description

replace a string at the positions specified by pos

Usage

```
## S3 method for class 'replace.at.pos'
str(str, pos, new, pos.mat.like.list = FALSE)
```

Arguments

`str`	a vector, or a single element
`pos`	a matrix of substring positions, or a list of such matrices if str is a vector
`new`	a vector of new strings for each substring position, or a list of such vectors if length(str)>1

Value

string (vector) of length(str) in which the substrings have been replaced

Examples

```
## Not run:
  str = "1234567890"
  pos = rbind(c(7,7),c(4,5))
  new = c("XXX","...")
  str.replace.at.pos(str,pos,new)

  str = c("1234567890","ahgdasdajsdgadhsabd")
  str.replace.at.pos(str,pos,new)

## End(Not run)
```

`str.replace.by.blocks` *Replaces in str every occurence of pattern by replacement*

Description

Replaces in str every occurence of pattern by replacement

Usage

```
## S3 method for class 'replace.by.blocks'
str(str, pattern, replacement, blocks = NULL,
  sub.txt = "SUB", block.start, block.end, block.ignore = NULL,
  use.levels = NULL, fixed = TRUE, only.replace.smaller.than = NULL,
  only.replace.larger.than = NULL)
```

Arguments

`str`	the string to replaced
`pattern`	the substring to be replaced
`replacment`	the new substrings
`block`	a block retrieved from str.block.pos alternatively, you can provide block.start and block.end
`block.start`	string with which the blocks start, e.g. "("
`block.end`	string with which the blocks end, e.g. ")"
`only.replace.smaller.than`	if not NULL only replaces matches whose number of characters is less or equal to only.replace.smaller.than
`only.replace.larger.than`	if not NULL only replaces matches whose number of characters is bigger or equal to only.replace.larger.than

Value

a string

Examples

```
## Not run:
  # Replace latex fractions
  str = "5+\frac{x^2+x^2}{1+\frac{2}{x*5}}*2"
  str.replace.by.blocks(str,"\frac{_SUB_}{_SUB_}","(_SUB1_)/(_SUB2_)",
                        block.start = "{", block.end = "}")
  str.replace.by.blocks(str,"\frac{_SUB_}{_SUB_}","(_SUB1_)/(_SUB2_)",
                        block.start = "{", block.end = "}",
                        only.replace.larger.than=20)
  str.replace.by.blocks(str,"\frac{_SUB_}{_SUB_}","(_SUB1_)/(_SUB2_)",
```

```
                        block.start = "{", block.end = "}",
                        only.replace.smaller.than=20)
    str ="-\frac{\sigma_{m}-\beta\sigma_{b}}{\beta-1}=\frac{\sigma_{m}-\beta\sigma_{b}}{1-\beta}"

     str ="\frac{1}{2}=\frac{3}{4}"
     str.replace.by.blocks(str,"\frac{_SUB_}{_SUB_}","(_SUB1_)/(_SUB2_)",
                        block.start = "{", block.end = "}")


   ## End(Not run)
```

`str.replace.list` *Performs sequentially all replacements of pattern and replace on the same strings str*

Description

A very slow implementation

Usage

```
## S3 method for class 'replace.list'
str(str, pattern, replacement, ...)
```

Examples

```
## Not run:
  str.replace.list("na dies ist doch",c("a","e"),c("A","E"))

## End(Not run)
```

`str.right` *keeps characters on right*

Description

keeps characters on right

Usage

```
## S3 method for class 'right'
str(str, len = 1)
```

str.right.of — *Returns the substring right to the first occurence of pattern*

Description

Returns the substring right to the first occurence of pattern

Usage

```
## S3 method for class 'right.of'
str(str, pattern, ..., not.found = str)
```

str.space — *Returns a string constisting of times spaces, vectorized over times*

Description

Returns a string constisting of times spaces, vectorized over times

Returns a string constisting of times spaces, vectorized over times

Usage

```
## S3 method for class 'space'
str(times, space = " ")

## S3 method for class 'space'
str(times, space = " ")
```

str.split — *Splits string vectors*

Description

Splits string vectors

Usage

```
## S3 method for class 'split'
str(str, pattern, first = FALSE, keep.match = FALSE, ...)
```

Arguments

str	a vector of strings
pattern	vector where splits take place

Value

A list with same length as str. Each list element i contains the split substrings from str[i]

Examples

```
## Not run:
  str = "Hi\n\nyou!"
  str.split(str,"\n", keep.match=!TRUE)

  str <- c("aes_afe_f", "qwe.rty", "yui0op[3", "b")
  #split x on the letter e
  str
  str.split(str, "e", keep.match=TRUE)
  str.split(str, "e", first=TRUE, keep.match=TRUE)

  str = c("aes_afe_fe")
  ignore.pos = cbind(1,3)
  str.split(str, "e", keep.match=TRUE, ignore.pos=ignore.pos)
  str.split(str, "e", first=TRUE,keep.match=TRUE, ignore.pos=ignore.pos)


  str = "abscde3823nsd34"
  str.split(str, "[a-z]*", fixed=FALSE, keep.match=TRUE)
  str.split(str, c("[a-z]*","d"), fixed=FALSE, keep.match=TRUE)

  str = c("abscde3823nsd34","8748274")
  str.split(str, c("[a-z]*","d"), fixed=FALSE, keep.match=TRUE)

## End(Not run)
```

`str.split.at.pos` *Splits a single string str at positions specified by pos*

Description

Splits a single string str at positions specified by pos

Usage

```
## S3 method for class 'split.at.pos'
str(str, pos, keep.pos = FALSE, compl = NULL,
  max.char = max(nchar(str)), pos.mat.like.list = FALSE)
```

Arguments

`str`	character vector that shall be splitted
`pos`	split positions can be vector: assuming an element of size 1 that specifies a single char at that positions n*2 matrix: first column left position, right column right position list of vectors or matrices, specifying different pos for different str

keep.pos default=FALSE. If TRUE add the tokens that describe the split to the result otherwise remove them

Value

single return is length of pos (if vector) or NCOL(pos) if pos is matrix

Examples

```
## Not run:
  str = c("1234567890")
  pos = c(3,5,7)
  str.split.at.pos(str,pos,keep.pos = FALSE)
  str.split.at.pos(str,pos,keep.pos = TRUE)
  pos = rbind(c(2,3),c(5,5),c(7,9))
  str.split.at.pos(str,pos,keep.pos = FALSE)
  str.split.at.pos(str,pos,keep.pos = TRUE)

  # Multiple str
  str = c("Hello ernie","abcg","hello erna")
  pos = c(2,5,8)
  str.split.at.pos(str,pos,keep.pos=TRUE)
  pos = list(c(3,5),c(2),c(1,9))
  str.split.at.pos(str,pos,keep.pos=TRUE)

  str = c("Hello ernie","abcdefg","hello erna")
  pos = str.locate.first(str,"e",ignore=ignore)
  pos
  str.split.at.pos(str,pos,keep.pos=TRUE,pos.mat.like.list=FALSE)
  str.split.at.pos(str,pos,keep.pos=TRUE,pos.mat.like.list=TRUE)


## End(Not run)
```

---

str.starts.with *Returns als elements of txt that begin with pattern*

Description

Returns als elements of txt that begin with pattern

Usage

```
## S3 method for class 'starts.with'
str(txt, pattern)
```

Examples

```
## Not run:
  str = c("Hi how are you", "hi", "now what", "Hi")
  str.starts.with(str,"Hi")

## End(Not run)
```

`str.tokenize` *An alternative interface to str.split*

Description

An alternative interface to str.split

Usage

```
## S3 method for class 'tokenize'
str(str, split = " ", only.one.split = FALSE,
  simplify = TRUE, ...)
```

`str.trim` *trims whitespaces from string*

Description

trims whitespaces from string

Usage

```
## S3 method for class 'trim'
str(txt)
```

to.char.matrix *converts into a vector of strings into a matrix of single characters*

Description

converts into a vector of strings into a matrix of single characters

Usage

```
to.char.matrix(str, drop = FALSE)
```

Examples

```
## Not run:
  str =c("Now that is a nice matrix","but short!")
  mat = to.char.matrix(str)
  mat
  char.matrix.to.str(mat)
  vec = to.char.vector(str,collapse="\n")
  vec
  char.vector.to.str(vec,collapse="\n")

## End(Not run)
```

to.char.vector *converts a string into a vector of single characters*

Description

converts a string into a vector of single characters

Usage

```
to.char.vector(str, collapse = "")
```

Index