Package: ribiosArg (via r-universe)

May 16, 2026

Type Package

Title Argument Handling for Command-Line, Stand-Alone R Scripts

Version 1.5.0

Date 2026-01-24

Description Provides functions to handle command-line arguments for R scripting. It enables building stand-alone R programs that accept and parse command-line options in 'BIOS' style. Zhang (2025) <https://github.com/bedapub/ribiosArg>.

Depends R (>= 3.4.0), ribiosUtils

LinkingTo ribiosUtils

Imports utils

Suggests testthat

License GPL-3

URL https://github.com/bedapub/ribiosArg

BugReports https://github.com/bedapub/ribiosArg/issues

Encoding UTF-8

LazyLoad yes

RoxygenNote 7.3.3

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

Date/Publication 2026-02-18 09:36:35 UTC

RemoteUrl https://github.com/bedapub/ribiosArg

RemoteRef HEAD

RemoteSha 677377544b839e542373d9319075ed0a1e162f57

Contents

`argGet` *Parse an argument*

Description

Get the value of an named argument

Usage

```
argGet(opt, default = NULL, choices = NULL)
```

Arguments

`opt`	name of the argument to be parsed
`default`	default values to be returned if the argument is not provided
`choices`	a character vector of accepted values; if a string outside the vector is provided, the function will stop and print error message

Details

The parsing is performed at C-level. It is an abbreiviation of argGetPos(opt, ind=1, default=NULL, choices=NULL)

Value

A character string representing the value of the argument

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

argParse, argGetPos, and argPresent

Examples

```
argGet("infile")
```

---

argGetPos *Parse an argument with the given position*

Description

Get the value of an named argument with the given position

Usage

```
argGetPos(opt, ind = 1L, default = NULL, choices = NULL)
```

Arguments

opt	name of the argument to be parsed
ind	index of the argument to be parsed, starting from 1.
default	default values to be returned if the argument is not provided
choices	a character vector of accepted values; if a string outside the vector is provided, the function will stop and print error message

Details

The parsing is performed at C-level. If the argument accepts only one value, users can also call argGet(opt, default=NULL, choices=NULL)

Value

A character string representing the value of the argument

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

argParse, argGet, and argPresent

Examples

```
argGetPos("thresholds", ind=2)
```

`argIsInit` *Check whether the argument parser has been initialized*

Description

Check whether the argument parser has been initialized

Usage

```
argIsInit()
```

Value

Logical, `TRUE` if `argParse` has been called, `FALSE` otherwise.

`argParse` *Parser of command-line parameters in BIOS style*

Description

Parser of command-line parameters in BIOS style

Test whether the given option is present in the command line or not

Usage

```
argParse(optargs, reqargs, usage = paste(scriptName(), "-h"), strict = TRUE)

argPresent(opt)
```

Arguments

`optargs`	String describing optional arguments. Syntax: `<optname1>[,paramcnt1] <optname2>[,paramcnt2]..` Example: "verbose outfile,1" means the command line has the syntax `prog [-verbose] [outfile name]`. It can be an empty string to express "no options". The value for `paramcnt` is 0.
`reqargs`	String describining required arguments. Syntax: `<argname1> <argname2>...`. Example: "infile outfile" means the command line has the syntax `prog [-infile ]infile [-outfile ]coutfile`. Even if it is empty, it is checked that at least one non-optional value is given.
`usage`	A character string to be printed if the command-line option parsing fails

strict	Logical, are extra un-prefixed parameters allowed? If set to TRUE, the un-prefixed parameters (which must be at the end of the command line) will be returned as a character vector.
opt	Character string, option name

Details

argParse must be called before argGet,argGetPos , argPresent, or argGetDefault. It checks whether the command line syntax agrees with the specification of optargs and reqargs. If not, the usage message is printed and the program exists.

argPresent returns a boolean value indicating whether the option is present or not.

If the syntax was found correct, argGetPos can be called to fetch the indth value of the option opt (indexing from 1). For instance, if the following option -ranges 3 5 is defined, argGetPos("range", 2) returns 5. argGet is a shortcut to fetch the first element. If the opt is missing, the default value will be returned.

Value

argParse is used for the side effects. If strict is set to TRUE, an invisible NULL is returned; otherwise, extra un-prefixed parameters are returned as an invisible character vector

argGet and argGetPos returns a character string. argPresent returns a boolean value.

In case of any error (wrong syntax, or not-existing option) the R session quits while printing the error message.

Examples

```
argParse("verbose threshold,2", "infile outfile",
        usage="prog [-infile ]infile [-outfile ]outfile [-verbose] [-threshold MIN MAX]")
argIsInit()
argPresent("verbose")
```

---

existArg *Test if named arguments exists*

Description

Test if named arguments exists

Usage

```
existArg(args)
```

Arguments

args	Argument names, without leading minus sign

Details

Options are those arguments with a leading minus sign (e.g. "-opt"). This function tells whether queried options exist in the argument list.

Value

A vector of logicals, indicating whether the arguments exist

See Also

getArg

Examples

```
comm <- paste(c("Rscript --vanilla -e", "'", "library(ribiosArg);",
             "existArg(c(\"opt\", \"opt2\", \"opt3\"))", "'",
              "-opt abc -opt3"), collapse=" ")
system(comm)
```

---

getArg *An R-implementation of getting named aguments*

Description

This function is out-dated. Please use `argparse` instead.

Usage

```
getArg(args, onlyArg = FALSE, missingArg = FALSE)
```

Arguments

`args`	Character strings, named arguments
`onlyArg`	Any type, What value should be returned if only the option is available and no value has been provided
`missingArg`	Any type, What value should be returned if the option is not available

Details

Options are those arguments with a leading minus sign. They can have one or more values following them, which will be taken as the value of the option. If no such values are availble, user could decide how to interpret the option by setting the `onlyArg` parameter. Similarly, missing options can be handled by `missingArg`

From version 1.0.3 `onlyArg` and `missingArg` accepts `NULL` as inputs.

Value

A list when more than one option were queried; or a vector if only one option was queried.

See Also

existArg

`isDebugging` *Test whether the environment is set for debugging*

Description

Test whether the environment is set for debugging

Usage

```
isDebugging()
```

Value

A logical value

See Also

setDebug and unsetDebug

Examples

```
isDebugging()
unsetDebug()
isDebugging()
setDebug()
```

`isIntDebugging` *Test whether the environment is set for debugging, or it's an interactive session*

Description

Test whether the environment is set for debugging, or it's an interactive session

Usage

```
isIntDebugging()
```

Value

A logical value

See Also

isDebugging

makeFactor *Make a factor*

Description

Make a factor

Usage

```
makeFactor(groups, levels = NULL, make.names = TRUE, verbose = FALSE)
```

Arguments

groups	Character strings
levels	Character vector, indicating strings
make.names	Should names be converted to adhere to the rule of variable names in R
verbose	Logical vector

Value

A factor with the specified levels.

Examples

```
makeFactor(c("A", "B", "C", "C", "A"), levels=LETTERS[3:1])
makeFactor(c("A 1", "B 2", "C 3", "C 3", "A 1"),
    levels=c("A 1", "C 3", "B 2"),
    make.names=TRUE)
makeFactor(c("A 1", "B 2", "C 3", "C 3", "A 1"),
    levels=c("A 1", "C 3", "B 2"),
    make.names=FALSE)
makeFactor(c("A 1", "B 2", "C 3", "C 3", "A 1"),
    levels=c("A 1", "C 3", "B 2"),
    make.names=FALSE, verbose=TRUE)
```

parseFactor *Parse a character string into factor*

Description

Parse a character string into factor

Usage

```
parseFactor(str, rlevels = NULL, make.names = TRUE, collapse = ",")
```

Arguments

str	A character string giving groups
rlevels	A character string giving levels
make.names	Logical, should names be converted to adhere to the rule of variable names in R
collapse	Character used in relevels to collapse different levels

Value

A factor parsed from the input string with the specified levels.

Examples

```
parseFactor("A,B,C,B,A", rlevels="A,B,C")

rgroup <- "A,B,C,D,B,C,A,D,B"
rlevels <- "D,A,B,C"
parseFactor(rgroup, rlevels)

groups <- c("ATest", "Control", "Control", "ATest")
levels <- c("Control", "ATest")
makeFactor(groups, levels)

# if 'groups' is a factor and 'levels' NULL or missing, its levels are respected
groups <- factor(c("B", "C", "A", "D"), levels=c("D","C","A","B"))
makeFactor(groups)


groups <- c("ATest", "Control", "Control", "ATest")
levels <- c("Control", "ATest", "Unknown")
makeFactor(groups, levels)

groups <- c("ATest", "Control", "Control", "ATest", "BTest")
levels <- c("Control", "ATest")
try(makeFactor(groups, levels))
```

---

parseFiles *Parse files from command-line options*

Description

Parse files from command line option, which can be (1) a string vector of files, (2) a file listing input files (e.g. pointer file), (3) a directory, or (4) a zip/tar/gz file (determined by suffix). In the later two cases, file patterns can be specified.

Usage

```
parseFiles(
  str,
  sep = ",",
  pattern = NULL,
  recursive = TRUE,
  ignore.case = TRUE
)
```

Arguments

str	A character string
sep	Seperator used in the string
pattern	Pattern string, if given, only files matching the pattern will be returned
recursive	In cse of directory or compressed files, whether files should be found recursively
ignore.case	In case of directory or compressed files, whether case should be ignored

Value

A character vector of file paths.

Note

In case of compressed files, a temp dir will be created: the user should take care of cleaning up!

---

parseNumVec *Parse a character string into a numveric vector*

Description

Numeric vectors can be given as arguments in two ways: (1) separated by blanks or (2) separated by other common separators, such as comma (,). This function parses a string, or a string vector into a numeric vector of expected length. In addition it is failure safe: user can specify the return value in case the parsing was not successful,

Usage

```
parseNumVec(str, expLen = 2, failVal = c(5, 5), sep = ",")
```

Arguments

str	A character string
expLen	Integer or NULL, Expected length of the numeric vector. When set to NULL, the numeric vector can be of variable length.
failVal	If the parsing failed (for example length not correct, or non-numeric values were provided, this value will be returned
sep	Separator in the character string, default ","

Details

The input value mostly comes from return values of the `argGet` function.

Value

A numeric vector of the parsed values, or `failVal` if parsing fails.

See Also

`argGet`

Examples

```
parseNumVec("3,7,9", expLen=3)
```

---

parsePairs *Parse key-value pairs from a character string*

Description

The function parses parameters in the form of `KEY1=VAL1,KEY2=VAL2,KEY3=VAL3` into `data.frame`.

Usage

```
parsePairs(
  str,
  collapse = ",",
  sep = "=",
  colnames = c("key", "value"),
  trim = TRUE,
  ...
)
```

Arguments

`str`	Character string
`collapse`	Collapse character used in the string
`sep`	Seperator used in the string
`colnames`	Column names of the returned `data.frame`
`trim`	Logical, whether additional spaces should be trimmed
`...`	Further parameters passed to `trim` for fine-tuning of trimming

Details

If input string is `NULL`, the function returns `NULL`. This can be useful in case the parameter is optional and not specified.

Value

A `data.frame` containing keys and values

See Also

`parseStrings`

Examples

```
parsePairs("A=3,B=4,C=5", collapse=",", sep="=")
parsePairs("A:3|B:4|C:5", collapse="|", sep=":")
```

`parseStrings` *Parse a character string into string vectors*

Description

This function parses collapsed multiple options into a vector of character strings. Each option is optionally trimmed of leading and tailing empty spaces given by `trim`. See examples.

Usage

```
parseStrings(str, collapse = ",", trim = TRUE, ...)
```

Arguments

`str`	A character string to be parsed
`collapse`	Character(s) used in the character string to concatenate strings
`trim`	Logical, whether additional spaces should be trimmed
`...`	Further parameters passed to `trim` for fine-tuning of trimming

Details

In case of multiple separators, they can be given by concatenating with piple signs, e.g. `,|\t`. If input string is `NULL`, the function returns `NULL`. This can be useful in case the parameter is optional and not specified.

Value

A vector of character strings

See Also

`strsplit`, `trim`

Examples

```
parseStrings("veni, vidi, vici")
parseStrings("veni, vidi, vici", trim=FALSE)
parseStrings("I came, I saw, I conquered")

# options are trimmed
parseStrings("a,b,\tc,d\n")
# it works also with only one option
parseStrings("a")
# more than one separators
parseStrings("a,b,c;d", collapse=",|;")
```

`ribiosArg` *ribiosIO ribiosIO provides Command-line argument handling for R scripting*

Description

ribiosIO ribiosIO provides Command-line argument handling for R scripting

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

`scriptInit` *Prepare the environment for a script*

Description

This function is called at the beginning of an Rscript, in order to prepare the R environment to run in a script setting.

Usage

```
scriptInit()
```

Value

No return value, called for side effects.

Examples

```
scriptInit()
```

`scriptName` *Returns the file name of the Rscript being executed*

Description

Get the file name of the Rscript that is currently being executed. The function is mainly called by stand-alone Rscripts.

Usage

```
scriptName()
```

Details

The name is determined by the `--file`/`-f` option in the command line.

When the R session was not initiated by a Rscript (i.e. there is no `--file` or `-f` option in the command line), `NULL` is returned.

Note that the function supports calling Rscript via `--file` or `-f` with `R`. This applies to cases where a Rscript, marked as executable, and is called from the command line.

Value

A character string containing the file name of the Rscript.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

`commandArgs` and `getArg`

Examples

```
scriptName()
```

`scriptPath` *Returns the path of the Rscript being executed*

Description

Get the normalised path of the Rscript that is currently being executed. The function is mainly called by stand-alone Rscripts.

Usage

```
scriptPath()
```

Details

The name is determined by the `--file`/`-f` option in the command line.

When the R session was not initiated by a Rscript (i.e. there is no `--file` or `-f` option in the command line), `NULL` is returned.

Note that the function supports calling Rscript via `--file` or `-f` with `R`. This applies to cases where a Rscript, marked as executable, and is called from the command line.

Value

A character string containing the normalised path of the Rscript.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

`scriptName`

Examples

```
scriptPath()
```

scriptSkeleton *Generate a Rscript with its skeleton*

Description

Generate a Rscript with its skeleton

Usage

```
scriptSkeleton(file)
```

Arguments

file	Output file. Use 'file=stdout()' to write the output to standard output.

Value

Invisibly returns the character vector of skeleton lines. Called for its side effect of writing to `file`.

Examples

```
scriptSkeleton(file = file.path(tempdir(), "myscript.R"))
```

setDebug *Set the enrivonment for debugging*

Description

Set the enrivonment for debugging

Usage

```
setDebug()
```

Value

A logical value, whether the setting was susccessful or not

See Also

`isDebugging` and `unsetDebug`

unsetDebug *Remove the debugging flag of the the enrivonment*

Description

Remove the debugging flag of the the enrivonment

Usage

```
unsetDebug()
```

Value

A logical value, whether the removal was successful or not

See Also

isDebugging and setDebug

Index