Package: scribe (via r-universe)

September 5, 2024

Title Command Argument Parsing

Version 0.3.0.9002

Maintainer Jordan Mark Barbone <jmbarbone@gmail.com>

Description A base dependency solution with basic argument parsing for use with 'Rscript'.

License MIT + file LICENSE

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Depends R (>= 3.6)

Imports methods, utils

Suggests covr, knitr, rmarkdown, spelling, testthat (>= 3.1.0), withr

Config/testthat/edition 3

VignetteBuilder knitr

URL https://jmbarbone.github.io/scribe/, https://github.com/jmbarbone/scribe

BugReports https://github.com/jmbarbone/scribe/issues

Config/testthat/parallel true

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

RemoteUrl https://github.com/jmbarbone/scribe

RemoteRef HEAD

RemoteSha 7fe5bc07bfeda0b071857a716d217f9351ce1e45

Contents

`command_args` *Command line arguments*

Description

Make a new scribeCommandArgs object

Usage

```
command_args(
  x = NULL,
  include = getOption("scribe.include", c("help", "version", NA_character_)),
  string = NULL,
  super = include
)
```

Arguments

`x`, `string`	Command line arguments; see `base::commandArgs()` for default. At least one parameter has to be `NULL`. When `string` is `NULL`, `x` is used, which defaults to `commandArgs(trailingOnly = TRUE)`. Otherwise the value of `x` is converted to a `character`. If `string` is not `NULL`, `scan()` will be used to split the value into a `character` vector.
`include`	Special default arguments to included. See `$initialize()` in scribeCommandArgs for more details.
`super`	When `TRUE` the scribeCommandArgs object will be initialized with standard *super* arguments (e.g., `---help`, `---version`)

Value

A scribeCommandArgs object

See Also

Other scribe: `new_arg()`, `scribeArg-class`, `scribeCommandArgs-class`

Examples

```
command_args()
command_args(c("-a", 1, "-b", 2))
command_args(string = "-a 1 -b 2")
```

`new_arg` *New command argument*

Description

Make a new scribeArg object

Usage

```
new_arg(
  aliases = "",
  action = arg_actions(),
  default = NULL,
  convert = scribe_convert(),
  n = NA_integer_,
  info = NULL,
  options = list(),
  stop = c("none", "hard", "soft"),
  execute = invisible
)
```

Arguments

`aliases`, `action`, `convert`, `options`, `default`, `info`, `n`, `stop`, `execute`
See `$initialize()` in scribeArg.

Value

A scribeArg object

See Also

Other scribe: `command_args()`, `scribeArg-class`, `scribeCommandArgs-class`

Examples

```
new_arg()
new_arg("values", action = "dots")
new_arg(c("-f", "--force"), action = "flag")
```

`scribeArg-class` *scribe argument*

Description

ReferenceClass object for managing arguments

Details

The scribeArg class sets specifications and controls for how command line arguments are to be parsed. These are meant to be used in conjunction with scribeCommandArgs and specifically with the Rscript utility. However, a use can define their own scribeArg separately.

Fields

`aliases [character]`
: A vector to denote the argument's name

`action [character]`
: An action for resolving the argument (see `default` for note on using another scribeArg object)

`default [ANY]`
: A default value. This can be another scribeArg object. When that is the case, the default value and action are pass through from the other scribeArg object.

`convert [ANY]`
: Passed to the `to` argument in `value_convert()`

`n [integer]`
: The length of the values

`info [character]`
: Additional information about the argument when printed

`options [list]`
: A named list of options (see **Options**)

`positional [logical]`
: Indicator if the argument is *positional* (i.e., not preceded by a `-` or `--` command line argument)

`resolved [logical]`
: Has the object been resolved

`value [ANY]`
: The resolve value

`stop [character]`
: `"none"`, `"hard"`, or `"soft"`

`execute [function]`
: (For advanced use). A `function` to be evaluated along with the arg. The function can have no parameters, a single parameter for the scribeArg object, or accept the scribeArg object as its first argument, and the scribeCommandArgs object as its second. Both objects will be passed by position

Methods

`get_action()` Retrieve action

`get_aliases()` Retrieve aliases

`get_default()` Retrieve the default value

`get_help()` Retrieve help information as a `character` vector

`get_name(clean = TRUE)` Retrieve names

`clean` When `TRUE` removes `-`s from text

`get_value()` Retrieve the resolved value

`help()` Print out formatted help information

`initialize( aliases = "", action = arg_actions(), default = NULL, convert = scribe_convert(), n = NA_intege`
Initialize the scribeArg object

See **fields** for parameter information.

`is_resolved()` Check if object has been resolved

Options

Several available options

`action="list" choices` An explicit set of values that argument must be. If the value parsed is not one of these, an error will occur.

`action="flag" no` When `TRUE` included appends `--no` to aliases to invert results
Example:
With the argument `new_arg("--test", options = list(no = TRUE))`, passing command arguments `--test` would set this to `TRUE` and `--no-test` explicitly set to `FALSE`.

See Also

Other scribe: `command_args()`, `new_arg()`, `scribeCommandArgs-class`

Examples

```
# new_arg() is recommended over direct use of scribeArg$new()

# arguments with `--` indicators
new_arg("--verbose", action = "flag")
new_arg(c("-f", "--force"), action = "flag")
new_arg("--values", action = "list")

# positional
new_arg("verbose", action = "flag")
new_arg("value", action = "list", n = 1)

# special `...` action which absorbs left-over arguments
new_arg("values", action = "dots", info = "list of values")
new_arg("...", info = "list of values") # defaults when alias is "..."
```

scribeCommandArgs-class *scribe command arguments*

Description

Reference class object for managing command line arguments.

Details

This class manages the command line argument inputs when passed via the Rscript utility. Take the simple script below which adds two numbers, which we will save in an executable file called `add.R`,

```
#!/usr/bin/env Rscript

library(scribe)
ca <- command_args()
ca$add_argument("--value1", default = 0L)
ca$add_argument("--value2", default = 0L)
args <- ca$parse()
writeLines(args$value1 + args$value2)
```

When called by a terminal, we can pass arguments and return a function.

```
add.R --value1 10 --value2 1
11
```

When testing, you can simulate command line arguments by passing them into the `input` field. By default, this will grab values from `base::commandArgs()`, so use with the Rscript utility doesn't require any extra steps.

Most methods are designed to return `.self`, or the scribeCommandArgs class. The exceptions to these are the the `$get_*()` methods, which return their corresponding values, and `$parse()` which returns a named `list` of the parsed input values.

Fields

`input [character]`
: A character vector of command line arguments. See also `command_args()`

`values [list]`
: A named `list` of values. Empty on initialization and populated during argument resolving.

`args [list]`
: A List of scribeArgs

`description [character]`
: Additional help information

`included [character]`
: Default scribeArgs to include

`examples [character]`
: Examples to print with help

`comments [character]`
: Comments printed with

`resolved [logical]`
: A `logical` value indicated if the `$resolve()` method has been successfully executed.

`working [character]`
: A copy of `input`. Note: this is used to track parsing progress and is not meant to be accessed directly.

`stop [character]`
: Determines parsing

`supers [list]`
: A list of `scribeSuperArgss`

Methods

`add_argument( ..., action = arg_actions(), default = NULL, convert = scribe_convert(), n = NA_integer_, in`
: Add a scribeArg to `args`

  `...` Either aliases or a scribeArg. If the latter, all other arguments are ignored. Note that only the first value (..1) is used.

  `action`, `options`, `convet`, `default`, `n`, `info`, `stop`, `execute` See `new_arg()`

`add_description(..., sep = "")` Add a value to `description`

  `...` Information to paste into the description

  `sep` `character` separate for `...`

`add_example(x, comment = "", prefix = "$ ")` Add a value to `examples`

  `x` A code example as a `character`

  `comment` An optional comment to append

  `prefix` An optional prefix for the example

`get_args(included = TRUE, super = FALSE)` Retrieve `args`

  `included` If `TRUE` also returns included default scribeArgs defined in `$initialize()`

  `super` If `TRUE` also returns super args

`get_description()` Retrieve `description`

`get_examples()` Retrieve `examples`

`get_input()` Retrieve `input`

`get_values(empty = FALSE, super = FALSE, included = FALSE)` Retrieve `values`

  `{empty}` If `TRUE` returns empty values, too

  `super` If `TRUE` also returns values from super args

  `included` If `TRUE` also returns included default scribeArgs defined in `$initialize()`

`help()` Print the help information

`initialize( input = "", include = c("help", "version", NA_character_), supers = include )`
Initialize the scribeCommandArgs object. The wrapper `command_args()` is recommended rather than calling this method directly.

`input` A `character` vector of command line arguments to parse

`include` A character vector denoting which default scribeArgs to include in `args`

`supers` A character vector denoting which default `scribeSuperArgs` to include in `supers` (i.e., arguments called with '—' prefixes)

`parse()` Return a named `list` of parsed values of from each scribeArg in `args`

`resolve()` Resolve the values of each scribeArg in `args`. This method is called prior to $parse()

`set_description(..., sep = "")` Set the value of `description`

`...` Information to paste into the description

`sep` `character` separate for `...`

`set_example(x = character(), comment = "", prefix = "$ ")` Set the value of `examples`

`x` A code example as a `character`

`comment` An optional comment to append

`prefix` An optional prefix for the example

`set_input(value)` Set `input`. Note: when called, `resolved` is (re)set to `FALSE` and values need to be parsed again.

`value` Value to set

`set_values(i = TRUE, value)` Set `values`

`i` Index value of `working` to set

`value` The value to set

`version()` Print the scribe-package version

See Also

Other scribe: `command_args()`, `new_arg()`, `scribeArg-class`

Examples

```
# command_args() is recommended over direct use of scribeCommandArgs$new()

ca <- command_args(c(1, 2, 3, "--verbose"))
ca$add_argument("--verbose", action = "flag")
ca$add_argument("...", "values", info = "values to add", default = 0.0)
args <- ca$parse()

if (args$verbose) {
  message("Adding ", length(args$values), " values")
}

sum(args$values)

# $parse() returns a named list, which means scribeCommandArgs can function
# as a wrapper for calling R functions inside Rscript
```

```
ca <- command_args(c("mean", "--size", 20, "--absolute"))
ca$add_argument("fun", action = "list")
ca$add_argument("--size", default = 5L)
ca$add_argument("--absolute", action = "flag")
args <- ca$parse()

my_function <- function(fun, size, absolute = FALSE) {
  fun <- match.fun(fun)
  x <- sample(size, size, replace = TRUE)
  res <- fun(x)
  if (absolute) res <- abs(res)
  res
}

do.call(my_function, args)
```

`value_convert` *Simple conversions*

Description

Convert character to data types

Usage

```
value_convert(x, to = default_convert)

scribe_convert(method = c("default", "evaluate", "none"))
```

Arguments

`x`	A vector of character values
`to`	What to convert `x` to (see details for more)
`method`	The conversion method: • `TRUE` or `"default"`: uses `value_convert()` • `"evaluate"` executes the string as an expression • `FALSE` or `NA` does nothing • When passed a `function`, simply returns the function

Details

`to` can be one of several values. Firstly the default of `default` calls several additional functions that attempt to resolve a transformation from a `character` vector to a different type. It is recommended for users to enter their own specifications instead. Secondly, a `function` (with a single argument) can be passed which will then be applied directly to `x`. Third, a *prototype* value can be passed. This might be risky for special types. Here, the values of `mode()`, `storage.mode()`, `attributes()`, and `class()` are captured and reassigned from `to` to `x`. A special check is implemented for `factors` to more safely convert. Lastly, `NULL` will do nothing and will simply return `x`.

Value

- `value_convert()`: A parsed value from `x`
- `scribe_convert()`: A function that takes a argument `x` and converts it

Examples

```
str(value_convert("2023-03-05", as.Date))
value_convert("a", factor(letters))
```

Index