Package 'optparse'

May 9, 2026

Encoding UTF-8

Type Package

Title Command Line Option Parser

Version 1.8.2

Description A command line parser inspired by Python's 'optparse' library to be used with Rscript to write ``#!'' shebang scripts that accept short and long flag/options.

License GPL (>= 2)

Copyright See file (inst/)COPYRIGHTS.

URL https://github.com/trevorld/r-optparse, https://trevorldavis.com/R/optparse/

BugReports https://github.com/trevorld/r-optparse/issues

LazyLoad yes

Depends R (>= 3.6.0)

Imports methods

Suggests knitr (>= 1.15.19), rmarkdown, stringr, testthat

Config/testthat/edition 3

VignetteBuilder knitr, rmarkdown

RoxygenNote 7.3.3

NeedsCompilation no

Author Trevor L. Davis [aut, cre] (ORCID: <https://orcid.org/0000-0001-6341-4639>),
Allen Day [aut] (Code and documentation ported from the getopt package.),
Python Software Foundation [ctb] (Some documentation from the optparse Python module.),
Steve Lianoglou [ctb],
Jim Nikelski [ctb],
Kirill Müller [ctb],
Peter Humburg [ctb],

Rich FitzJohn [ctb],
Gyu Jin Choi [ctb]

Maintainer Trevor L. Davis <trevor.l.davis@gmail.com>

Repository CRAN

Date/Publication 2026-04-17 07:10:02 UTC

Contents

`optparse-package` *Command line option parser*

Description

Goal is to create an R package of a command line parser inspired by Python's "optparse" library.

Details

`optparse` is primarily intended to be used with "Rscript". It facilitates writing "#!" shebang scripts that accept short and long flags/options. It can also be used from directly, but is probably less useful in this context.

See package vignette for a more detailed example.

Notes on naming convention in package: 1. An option is one of the shell-split input strings. 2. A flag is a type of option. a flag can be defined as having no argument (defined below), a required argument, or an optional argument. 3. An argument is a type of option, and is the value associated with a flag. 4. A long flag is a type of flag, and begins with the string "–". If the long flag has an associated argument, it may be delimited from the long flag by either a trailing =, or may be the subsequent option. 5. A short flag is a type of flag, and begins with the string "-". If a short flag has an associated argument, it is the subsequent option. short flags may be bundled together, sharing a single leading ""-"", but only the final short flag is able to have a corresponding argument. %%%

Author(s)

Trevor L. Davis.

Some documentation and unit tests ported from Allen Day's getopt package.

The documentation for Python's optparse library, which this package is based on, is Copyright 1990-2009, Python Software Foundation.

References

Python's optparse library, which this package is based on, is described here: https://docs.python.org/3/library/optparse.html

See Also

getopt

Examples

```
example_file <- system.file("exec", "example.R", package = "optparse")
example_file_2 <- system.file("exec", "display_file.R", package = "optparse")
## Not run:
    readLines(example_file)
    readLines(example_file_2)

## End(Not run)
```

IndentedHelpFormatter *Builtin help text formatters*

Description

IndentedHelpFormatter() is the default help text formatter. TitledHelpFormatter() is an alternative help text formatter.

Usage

```
IndentedHelpFormatter(object)

TitledHelpFormatter(object)
```

Arguments

object	An OptionParser() object.

Value

NULL invisibly. As a side effect prints out help text.

Examples

```
parser <- OptionParser(formatter = IndentedHelpFormatter)
parser <- add_option(parser, "--generator", help = "Generator option")
parser <- add_option(parser, "--count", help = "Count option")
print_help(parser)

parser <- OptionParser(formatter = TitledHelpFormatter)
```

```
parser <- add_option(parser, "--generator", help = "Generator option")
parser <- add_option(parser, "--count", help = "Count option")
print_help(parser)
```

---

make_option *Functions to enable our OptionParser to recognize specific command line options.*

Description

add_option() adds a option to a prexisting OptionParser instance whereas make_option() is used to create a list of OptionParserOption instances that will be used in the option_list argument of the OptionParser function to create a new OptionParser instance.

Usage

```
make_option(
  opt_str,
  action = NULL,
  type = NULL,
  dest = NULL,
  default = NULL,
  help = "",
  metavar = NULL,
  callback = NULL,
  callback_args = NULL,
  const = NULL,
  required = FALSE
)

add_option(
  object,
  opt_str,
  action = NULL,
  type = NULL,
  dest = NULL,
  default = NULL,
  help = "",
  metavar = NULL,
  callback = NULL,
  callback_args = NULL,
  const = NULL,
  required = FALSE
)
```

Arguments

opt_str	A character vector containing the string of the desired long flag comprised of -- followed by a non-dash character (and optionally more characters), and optionally a string of the desired short flag comprised of the - followed by a single non-dash character. We don't allow = or whitespace characters in flags.
action	A character string describing the action optparse should take when it encounters an option. One of: • "append": appends each occurrence's value to default (or to an empty vector if default is NULL). Returns NULL if never seen and default is NULL. • "callback": stores the return value of the callback function. • "count": counts the number of times the flag is seen and adds it to default (treated as 0L if not supplied). Returns NULL if never seen and no default was supplied. • "store" (default): stores the specified following value. • "store_const": stores const if the flag is seen, otherwise default. Returns NULL if the flag is not seen and default is NULL. • "store_true": stores TRUE if the option is found. • "store_false": stores FALSE if the option is found. If callback is not NULL the default action is "callback" otherwise it is "store".
type	A character string specifying which data type to store: "logical", "integer", "double", "complex", or "character" ("numeric" is an alias for "double"). Defaults: • if action == "count" then "integer" • if action %in% c("store_false", "store_true") then "logical" • if action == "store" and default is not NULL then typeof(default)else if default is NULL then "character"
dest	A character string specifying what field in the list returned by parse_args() should optparse store the option value. Default is derived from the long flag in opt_str.
default	The default value optparse should use if it does not find the option on the command line.
help	A character string describing the option, used by print_help() in generating a usage message. "%default" will be substituted by the value of default.
metavar	A character string that stands in for the option argument when printing help text. Default is the value of dest.
callback	A function that executes after the option value is fully parsed. Its return value is assigned to the option. Arguments are: the option S4 object, the long flag string, the value of the option, the parser S4 object, and ....
callback_args	A list of additional arguments passed to callback (via do.call()).
const	The value to store when action = "store_const" and the flag is seen. Ignored for all other actions.

required	If TRUE, parse_args() will throw an error if this option is not provided on the command line. **Note:** Required options are generally considered bad form because users expect *options* to be *optional*, and thus they should be avoided when possible.
object	An instance of the OptionParser class

Value

Both make_option() and add_option() return instances of class OptionParserOption.

Errors

The following classed errors may be thrown:

- optparse_option_error: invalid option definition (bad flag string, unrecognized action, etc.).
  - optparse_option_conflict_error: duplicate flag detected when adding an option to a parser.

References

Python's optparse library, which inspires this package, is described here: https://docs.python.org/3/library/optparse.html

See Also

parse_args() OptionParser()

Examples

```
make_option("--longflag")
make_option(c("-l", "--longflag"))
make_option("--integer", type = "integer", default = 5)
make_option("--integer", default = as.integer(5))  # same as previous

# examples from package vignette
make_option(c("-v", "--verbose"), action = "store_true", default = TRUE,
    help = "Print extra output [default]")
make_option(c("-q", "--quietly"), action = "store_false",
  dest = "verbose", help = "Print little output")
make_option(c("-c", "--count"), type = "integer", default = 5,
    help = "Number of random normals to generate [default %default]",
    metavar = "number")
make_option("--generator", default = "rnorm",
    help = "Function to generate random deviates [default \"%default\"]")
make_option("--mean", default = 0,
    help = "Mean if generator == \"rnorm\" [default %default]")
make_option("--sd", default = 1, metavar = "standard deviation",
    help = "Standard deviation if generator == \"rnorm\" [default %default]")
```

`OptionParser` *A function to create an instance of a parser object*

Description

This function is used to create an instance of a parser object which when combined with the `parse_args()`, `make_option()`, and `add_option()` methods is very useful for parsing options from the command line.

Usage

```
OptionParser(
  usage = "usage: %prog [options]",
  option_list = list(),
  add_help_option = TRUE,
  prog = NULL,
  description = "",
  epilogue = "",
  formatter = IndentedHelpFormatter
)
```

Arguments

`usage`	The program usage message that will printed out if `parse_args()` finds a help option, `\%prog` is substituted with the value of the `prog` argument.
`option_list`	A list of of `OptionParserOption` instances that will define how `parse_args()` reacts to command line options. `OptionParserOption` instances are usually created by `make_option()` and can also be added to an existing `OptionParser` instance via the `add_option()` function.
`add_help_option`	Whether a standard help option should be automatically added to the `OptionParser` instance.
`prog`	Program name to be substituted for `\%prog` in the usage message (including description and epilogue if present), the default is to use the actual Rscript file name if called by an Rscript file and otherwise keep `\%prog`.
`description`	Additional text for `print_help()` to print out between usage statement and options statement
`epilogue`	Additional text for `print_help()` to print out after the options statement
`formatter`	A function that formats usage text. The function should take only one argument (an `OptionParser()` object). Default is `IndentedHelpFormatter()`. The other builtin formatter provided by this package is `TitledHelpFormatter()`.

Value

An instance of the `OptionParser` class.

References

Python's optparse library, which inspired this package, is described here: https://docs.python.org/3/library/optparse.html

See Also

parse_args() make_option() add_option()

OptionParser-class *Option Parser*

Description

Option Parser

Slots

usage The program usage message that will printed out if parse_args() finds a help option, \%prog is substituted with the value of the prog argument.

options A list of of OptionParserOption instances that will define how parse_args() reacts to command line options. OptionParserOption instances are usually created by make_option() and can also be added to an existing OptionParser instance via the add_option() function.

description Additional text for print_help() to print out between usage statement and options statement

epilogue Additional text for print_help() to print out after the options statement

formatter A function that print_help() will use to print out after the options statement. Default is IndentedHelpFormatter(). This package also provides the builtin formatter TitledHelpFormatter().

See Also

OptionParserOption

OptionParserOption-class
Class to hold information about command-line options

Description

Class to hold information about command-line options

Slots

`short_flag` String of the desired short flag comprised of the `-` followed by a single non-dash character (but not `=` or a whitespace character).

`long_flag` String of the desired long flag comprised of `--` followed by a non-dash character and then (optionally) more characters (but not any `=` or whitespace characters).

`action` A character string describing the action `optparse` should take when it encounters an option. One of:

- `"append"`: appends each occurrence's value to `default` (or to an empty vector if `default` is `NULL`). Returns `NULL` if never seen and `default` is `NULL`.
- `"callback"`: stores the return value of the `callback` function.
- `"count"`: counts the number of times the flag is seen and adds it to `default` (treated as `0L` if not supplied). Returns `NULL` if never seen and no `default` was supplied.
- `"store"` (default): stores the specified following value.
- `"store_const"`: stores `const` if the flag is seen, otherwise `default`. Returns `NULL` if the flag is not seen and `default` is `NULL`.
- `"store_true"`: stores `TRUE` if the option is found.
- `"store_false"`: stores `FALSE` if the option is found.

If `callback` is not `NULL` the default action is `"callback"` otherwise it is `"store"`.

`type` A character string specifying which data type to store: `"logical"`, `"integer"`, `"double"`, `"complex"`, or `"character"` (`"numeric"` is an alias for `"double"`). Defaults:

- if `action == "count"` then `"integer"`
- if `action %in% c("store_false", "store_true")` then `"logical"`
- if `action == "store"` and `default` is not `NULL` then `typeof(default)`else if `default` is `NULL` then `"character"`

`dest` A character string specifying what field in the list returned by `parse_args()` should `optparse` store the option value. Default is derived from the long flag in `opt_str`.

`default` The default value `optparse` should use if it does not find the option on the command line.

`const` The value to store when `action = "store_const"` and the flag is seen. Ignored for all other actions.

`required` If `TRUE`, `parse_args()` will throw an error if this option is not provided on the command line. **Note:** Required options are generally considered bad form because users expect *options* to be *optional*, and thus they should be avoided when possible.

`help` A character string describing the option, used by `print_help()` in generating a usage message. `"%default"` will be substituted by the value of `default`.

`metavar` A character string that stands in for the option argument when printing help text. Default is the value of `dest`.

`callback` A function that executes after the option value is fully parsed. Its return value is assigned to the option. Arguments are: the option S4 object, the long flag string, the value of the option, the parser S4 object, and `...`.

`callback_args` A list of additional arguments passed to `callback` (via `do.call()`).

See Also

`make_option()`

parse_args *Parse command line options.*

Description

parse_args() parses command line options using an OptionParser instance for guidance. parse_args2() is a wrapper to parse_args() setting the options positional_arguments and convert_hyphens_to_underscores to TRUE.

Usage

```
parse_args(
  object,
  args = commandArgs(trailingOnly = TRUE),
  print_help_and_exit = TRUE,
  positional_arguments = FALSE,
  convert_hyphens_to_underscores = FALSE
)

parse_args2(
  object,
  args = commandArgs(trailingOnly = TRUE),
  print_help_and_exit = TRUE
)
```

Arguments

object	An OptionParser instance.
args	A character vector containing command line options to be parsed. Default is everything after the Rscript program in the command line. If positional_arguments is not FALSE then parse_args() will look for positional arguments at the end of this vector.
print_help_and_exit	Whether parse_args() should call print_help() to print out a usage message and exit the program. Default is TRUE.
positional_arguments	Number of *positional* arguments. A numeric denoting the exact number of supported arguments, or a numeric vector of length two denoting the minimum and maximum number of arguments (Inf for no limit). The value TRUE is equivalent to c(0, Inf). The default FALSE is supported for backward compatibility only, as it alters the format of the return value.
convert_hyphens_to_underscores	If the names in the returned list of options contains hyphens then convert them to underscores. The default FALSE is supported for backward compatibility reasons as it alters the format of the return value

Value

Returns a list with field `options` containing our option values as well as another field `args` which contains a vector of positional arguments. For backward compatibility, if and only if `positional_arguments` is `FALSE`, returns a list containing option values.

Errors

The following classed errors may be thrown:

- `optparse_parse_error`: base class for all parse-time errors.
  - `optparse_bad_option_error`: unrecognized, misused, or argument-requiring option.
    - `optparse_ambiguous_option_error`: ambiguous abbreviated long flag.
  - `optparse_bad_positional_arguments_error`: wrong number of positional arguments supplied.
  - `optparse_missing_required_error`: a required option was not supplied.

References

Python's optparse library, which inspired this package, is described here: https://docs.python.org/3/library/optparse.html

See Also

`OptionParser()` `print_help()`

Examples

```
# example from vignette
option_list <- list(
   make_option(c("-v", "--verbose"), action = "store_true", default = TRUE,
       help = "Print extra output [default]"),
   make_option(c("-q", "--quietly"), action = "store_false",
       dest = "verbose", help = "Print little output"),
   make_option(c("-c", "--count"), type = "integer", default = 5,
       help = "Number of random normals to generate [default %default]",
       metavar = "number"),
   make_option("--generator", default = "rnorm",
       help = "Function to generate random deviates [default \"%default\"]"),
   make_option("--mean", default = 0,
       help = "Mean if generator == \"rnorm\" [default %default]"),
   make_option("--sd", default = 1, metavar = "standard deviation",
       help = "Standard deviation if generator == \"rnorm\" [default %default]")
   )
parse_args(OptionParser(option_list = option_list), args = c("--sd=3", "--quietly"))

# example from vignette using positional arguments
option_list2 <- list(
   make_option(c("-n", "--add-numbers"), action = "store_true", default = FALSE,
       help = "Print line number at the beginning of each line [default]")
   )
```

```
parser <- OptionParser(usage = "%prog [options] file", option_list = option_list2)

parse_args(parser, args = c("--add-numbers", "example.txt"), positional_arguments = TRUE)

parse_args(parser, args = c("--add-numbers", "example.txt"), positional_arguments = TRUE,
         convert_hyphens_to_underscores = TRUE)

parse_args2(parser, args = c("--add-numbers", "example.txt"))
```

`print_help` *Printing an usage message from an OptionParser object*

Description

`print_help()` print an usage message from an OptionParser object, usually called by `parse_args()` when it encounters a help option.

Usage

```
print_help(object)
```

Arguments

`object`	A `OptionParser` instance.

Value

`print_help()` uses the `cat` function to print out a usage message. It returns `invisible(NULL)`.

References

Python's `optparse` library, which inspired this package, is described here: https://docs.python.org/3/library/optparse.html

See Also

`parse_args()` `OptionParser()`

Index