Package 'argparse'

October 31, 2026

Type Package

Title Command Line Optional and Positional Argument Parser

Version 2.3.3

Description A command line parser to
be used with 'Rscript' to write ``#!'' shebang scripts that gracefully
accept positional and optional arguments and automatically generate usage.

License GPL (>= 2)

Copyright See file (inst/)COPYRIGHTS.

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

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

LazyLoad yes

Depends R (>= 3.6.0)

Imports findpython, jsonlite, R6

SystemRequirements python (>= 3.2)

Suggests knitr (>= 1.15.19), rmarkdown, testthat

VignetteBuilder knitr

Encoding UTF-8

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Trevor L. Davis [aut, cre] (ORCID:
<https://orcid.org/0000-0001-6341-4639>),
Allen Day [ctb] (Some documentation and examples ported from the getopt
package.),
Python Software Foundation [ctb] (Some documentation from the argparse
Python module.),
Paul Newell [ctb]

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

Repository CRAN

Date/Publication 2026-10-31 07:10:02 UTC

Contents

ArgumentGroup 2
ArgumentParser 3
MutuallyExclusiveGroup 4
Parser 5
Subparsers 9

Index **11**

`ArgumentGroup` *Argument group*

Description

An `ArgumentGroup` R6 object wraps a Python `argparse` argument group. Objects of this class are returned by the `add_argument_group()` method of Parser objects. Argument groups are used to group related arguments together in the help message.

Super class

`Group` -> `ArgumentGroup`

Methods

Public methods:

- `ArgumentGroup$add_mutually_exclusive_group()`

`ArgumentGroup$add_mutually_exclusive_group()`**:** Create a mutually exclusive group. Only one of the arguments added to the group may be present on the command line.

Usage:

```
ArgumentGroup$add_mutually_exclusive_group(required = FALSE)
```

Arguments:

`required` If `TRUE` then exactly one of the arguments in the group must be present on the command line.

Returns: A MutuallyExclusiveGroup object.

References

https://docs.python.org/3/library/argparse.html#argument-groups

See Also

`ArgumentParser()`, Parser, MutuallyExclusiveGroup

Examples

```
if (argparse:::detects_python()) {
  parser <- ArgumentParser(prog = "PROG", add_help = FALSE)
  group <- parser$add_argument_group("group")
  group$add_argument("--foo", help = "foo help")
  group$add_argument("bar", help = "bar help")
  parser$print_help()
}
```

`ArgumentParser` *Create a command line parser*

Description

`ArgumentParser()` creates a parser object that acts as a wrapper to Python's `argparse` module

Usage

```
ArgumentParser(..., python_cmd = NULL)
```

Arguments

`...`	Arguments cleaned and passed to Pythons `argparse.ArgumentParser()`
`python_cmd`	Python executable for `argparse` to use. Must have `argparse` and `json` modules (automatically bundled with Python 2.7 and 3.2+). If you need Unicode argument support then you must use Python 3.0+. Default will be to use `findpython` package to find suitable Python binary.

Value

`ArgumentParser()` returns a Parser R6 object which contains an `add_argument()` function to add arguments to the parser, a `parse_args()` function to parse command line arguments into a list, a `print_help()` and a `print_usage()` function to print usage information. See code examples, package vignette, and corresponding python module for more information on how to use it.

References

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

See Also

Parser, ArgumentGroup, MutuallyExclusiveGroup, and Subparsers for documentation of the R6 objects returned by `ArgumentParser()` and its methods.

Examples

```
run_example <- argparse:::detects_python()
if (run_example) {
  parser <- ArgumentParser(description='Process some integers')
  parser$add_argument('integers', metavar='N', type = "integer", nargs='+',
                      help='an integer for the accumulator')
  parser$add_argument('--sum', dest='accumulate', action='store_const',
                      const='sum', default='max',
                      help='sum the integers (default: find the max)')
  parser$print_help()
}
if (run_example) {
  # default args for ArgumentParser()$parse_args are commandArgs(TRUE)
  # which is what you'd want for an Rscript but not for interactive use
  args <- parser$parse_args(c("--sum", "1", "2", "3"))
  accumulate_fn <- get(args$accumulate)
  print(accumulate_fn(args$integers))
}
```

MutuallyExclusiveGroup

Mutually exclusive argument group

Description

A `MutuallyExclusiveGroup` R6 object wraps a Python `argparse` mutually exclusive group. Objects of this class are returned by the `add_mutually_exclusive_group()` method of Parser and ArgumentGroup objects. Only one of the arguments added to a mutually exclusive group may be present on the command line.

Unlike an ArgumentGroup, a mutually exclusive group can't contain nested groups (Python 3.14 and later no longer allow this).

Super class

`Group` -> `MutuallyExclusiveGroup`

References

https://docs.python.org/3/library/argparse.html#mutual-exclusion

See Also

`ArgumentParser()`, Parser, ArgumentGroup

Examples

```
if (argparse:::detects_python()) {
  parser <- ArgumentParser(prog = "PROG")
  group <- parser$add_mutually_exclusive_group()
  group$add_argument("--foo", action = "store_true")
  group$add_argument("--bar", action = "store_false")
  parser$parse_args("--foo")
}
```

`Parser` *Argument parser*

Description

A `Parser` R6 object wraps a Python `argparse` argument parser. Objects of this class are returned by `ArgumentParser()` and by the `add_parser()` method of Subparsers objects.

If a parse error occurs (or help or version information is requested) then in an interactive session an error is raised whereas in a non-interactive session (i.e. a script) a message is printed and R exits.

Methods

Public methods:

- `Parser$parse_args()`
- `Parser$parse_intermixed_args()`
- `Parser$parse_known_args()`
- `Parser$parse_known_intermixed_args()`
- `Parser$format_help()`
- `Parser$format_usage()`
- `Parser$print_help()`
- `Parser$print_usage()`
- `Parser$get_default()`
- `Parser$set_defaults()`
- `Parser$add_argument()`
- `Parser$add_argument_group()`
- `Parser$add_mutually_exclusive_group()`
- `Parser$add_subparsers()`
- `Parser$new()`

`Parser$parse_args()`**:** Parse command line arguments.

Usage:

```
Parser$parse_args(args = commandArgs(TRUE))
```

Arguments:

`args` A character vector of command line arguments.

Returns: A named list of argument values.

`Parser$parse_intermixed_args()`: Parse command line arguments allowing optional and positional arguments to be intermixed.

Usage:

```
Parser$parse_intermixed_args(args = commandArgs(TRUE))
```

Arguments:

`args` A character vector of command line arguments.

Returns: A named list of argument values.

`Parser$parse_known_args()`: Parse command line arguments without raising an error for unrecognized arguments.

Usage:

```
Parser$parse_known_args(args = commandArgs(TRUE))
```

Arguments:

`args` A character vector of command line arguments.

Returns: A list of length two. The first element is a named list of argument values. The second element is a character vector of the remaining unrecognized arguments.

`Parser$parse_known_intermixed_args()`: Parse command line arguments allowing optional and positional arguments to be intermixed and without raising an error for unrecognized arguments.

Usage:

```
Parser$parse_known_intermixed_args(args = commandArgs(TRUE))
```

Arguments:

`args` A character vector of command line arguments.

Returns: A list of length two. The first element is a named list of argument values. The second element is a character vector of the remaining unrecognized arguments.

`Parser$format_help()`: Format the help message.

Usage:

```
Parser$format_help()
```

Returns: A string containing the help message.

`Parser$format_usage()`: Format the usage message.

Usage:

```
Parser$format_usage()
```

Returns: A string containing the usage message.

`Parser$print_help()`: Print the help message.

Usage:

```
Parser$print_help()
```

Returns: NULL (invisibly).

`Parser$print_usage()`: Print the usage message.

Usage:

```
Parser$print_usage()
```

Returns: NULL (invisibly).

`Parser$get_default()`: Not currently supported. Always raises an error.

Usage:

```
Parser$get_default(...)
```

Arguments:

... Ignored.

`Parser$set_defaults()`: Set parser-level default values for arguments.

Usage:

```
Parser$set_defaults(...)
```

Arguments:

... Named arguments cleaned and passed to Python's `set_defaults()` method.

Returns: NULL (invisibly).

`Parser$add_argument()`: Define how a single command line argument should be parsed.

Usage:

```
Parser$add_argument(...)
```

Arguments:

... Arguments cleaned and passed to Python's `add_argument()` method. The `type` argument may be one of `"logical"`, `"integer"`, `"double"`, `"numeric"`, or `"character"`.

Returns: NULL (invisibly).

`Parser$add_argument_group()`: Create an argument group. Arguments added to the group are displayed together in the help message.

Usage:

```
Parser$add_argument_group(...)
```

Arguments:

... Arguments cleaned and passed to Python's `add_argument_group()` method such as `title` and `description`.

Returns: An ArgumentGroup object.

`Parser$add_mutually_exclusive_group()`: Create a mutually exclusive group. Only one of the arguments added to the group may be present on the command line.

Usage:

```
Parser$add_mutually_exclusive_group(required = FALSE)
```

Arguments:

`required` If `TRUE` then exactly one of the arguments in the group must be present on the command line.

Returns: A MutuallyExclusiveGroup object.

`Parser$add_subparsers()`**:** Add support for sub-commands.

Usage:

```
Parser$add_subparsers(...)
```

Arguments:

`...` Arguments cleaned and passed to Python's `add_subparsers()` method such as `title`, `description`, `help`, `dest`, and `required`.

Returns: A Subparsers object.

`Parser$new()`**:** Create a new `Parser` object. Users should not call this directly but instead use `ArgumentParser()`.

Usage:

```
Parser$new(python_code, name)
```

Arguments:

`python_code` An internal object that accumulates the Python code used to build the parser.

`name` Name of the Python variable holding this object.

References

Python's `argparse` documentation: https://docs.python.org/3/library/argparse.html

See Also

`ArgumentParser()`, ArgumentGroup, MutuallyExclusiveGroup, Subparsers

Examples

```
run_example <- argparse:::detects_python()
if (run_example) {
  parser <- ArgumentParser(description = "Process some integers")
  parser$add_argument("integers", metavar = "N", type = "integer",
                      nargs = "+", help = "an integer for the accumulator")
  parser$add_argument("--sum", dest = "accumulate", action = "store_const",
                      const = "sum", default = "max",
                      help = "sum the integers (default: find the max)")
  parser$print_usage()
}
if (run_example) {
  args <- parser$parse_args(c("--sum", "1", "2", "3"))
  print(args)
}
if (run_example) {
  args <- parser$parse_known_args(c("--sum", "1", "2", "--unknown"))
  print(args)
}
```

Subparsers *Subparsers*

Description

A Subparsers R6 object wraps the Python argparse object used to add sub-commands. Objects of this class are returned by the add_subparsers() method of Parser objects.

Methods

Public methods:

- Subparsers$new()
- Subparsers$add_parser()

Subparsers$new(): Create a new Subparsers object. Users should not call this directly but instead use the add_subparsers() method of a Parser object.

Usage:

```
Subparsers$new(python_code, name)
```

Arguments:

python_code An internal object that accumulates the Python code used to build the parser.

name Name of the Python variable holding this object.

Subparsers$add_parser(): Add a sub-command parser.

Usage:

```
Subparsers$add_parser(...)
```

Arguments:

... Arguments cleaned and passed to Python's add_parser() method. The first argument is usually the name of the sub-command. Accepts the same arguments as ArgumentParser() as well as help and aliases.

Returns: A Parser object.

References

https://docs.python.org/3/library/argparse.html#sub-commands

See Also

ArgumentParser(), Parser

Examples

```
if (argparse:::detects_python()) {
  parser <- ArgumentParser(prog = "PROG")
  parser$add_argument("--foo", action = "store_true", help = "foo help")
  subparsers <- parser$add_subparsers(help = "sub-command help")
  parser_a <- subparsers$add_parser("a", help = "a help")
  parser_a$add_argument("bar", type = "integer", help = "bar help")
  parser$parse_args(c("a", "12"))
}
```

Index

ArgumentGroup, 2, *3*, *4*, *7*, *8*
ArgumentParser, 3
ArgumentParser(), *2*, *4*, *5*, *8*, *9*

Group, *2*, *4*

MutuallyExclusiveGroup, *2*, *3*, 4, *8*

Parser, *2–4*, 5, *9*

Subparsers, *3*, *5*, *8*, 9