Package ‘defineOptions’

October 28, 2023

Type Package

Title Define and Parse Command Line Options

Version 0.9

Date 2023-10-21

Maintainer Toshihiro Umehara <toshi@niceume.com>

Description Parses command line arguments and supplies values to scripts. Users can specify names to which parsed inputs are assigned, value types into which inputs are cast, long options or short options, input splitters and callbacks that define how options should be specified and how input values are supplied.

Imports methods

Suggests RUnit

License GPL (>= 3)

URL https://github.com/niceume/defineOptions

BugReports https://github.com/niceume/defineOptions

NeedsCompilation no

Author Toshihiro Umehara [aut, cre]

Repository CRAN

Date/Publication 2023-10-28 15:20:02 UTC

R topics documented:

defineOptions-package *Define and Parse Command Line Options*

Description

Parses command line arguments and supplies values to scripts. Users can specify names to which parsed inputs are assigned, value types into which inputs are cast, long options or short options, input splitters and callbacks that define how options should be specified and how input values are supplied.

Details

Definitions are consturcted by calling `define_option` method for `ParserDef` object, which is instantiated by `new_parser_def` function. The second argument of `define_option` takes a list that has defition about how to parse and store its option value. The definition also holds information about how to behave when the option is not specified. Finally, `parse_with_defs` function takes command line arguments and `ParserDef` object and returns parsing result.

Author(s)

NA Maintainer: Toshihiro Umehara <toshi@niceume.com>

See Also

`ParserDef` `new_parser_def` `define_option` `parse_with_defs` `callbacks`

Examples

```
library(defineOptions)
parser_def = new_parser_def() |>
    define_option(
        list(
            def_name = "target_range",
            def_type = "integer",
            long_option = "--target-range",
            short_option = "-r",
            input_splitter = ",",
            callback = opt_optional_input_required( input_when_omitted = "70,180" )
        )
    ) |>
    define_option(
        list(
            def_name = "exclude_weekend",
            def_type = "logical",
            long_option = "--exclude-weekend",
            callback = opt_optional_input_disallowed( input_when_specified = "TRUE",
                                                      input_when_omitted = "FALSE" )
        )
    )|>
    define_option(
        list(
            def_name = "output_path",
            def_type = "character",
            long_option = "--output",
            callback = opt_required_input_required()
        )
    )

# In practice, command line arguments can be obtained by commandArgs() function
# with trailingOnly option TRUE.
# command_arguments = commandArgs(trailingOnly = TRUE)

example_string = "input1.txt input2.txt --target-range 60,140 --exclude-weekend --output log.data"
command_arguments = strsplit( example_string, " ")[[1]]

parsed_args = parse_with_defs( parser_def, command_arguments)
print(parsed_args)
```

`Built-in callbacks for option definitions` *Built-in callbacks for option definitions*

Description

define_option function takes an callback argument. The following functions return built-in callbacks for the callback argument.

Usage

```
opt_optional_input_required( input_when_omitted )
opt_optional_input_disallowed( input_when_specified, input_when_omitted)
opt_required_input_required()
```

Arguments

`input_when_omitted`	character
`input_when_specified`	character

Details

opt_optional_input_required() function returns a callback that is used to define that the option is optional but when the option is specified its input value is required to be specified. opt_optional_input_disallowed() function returns a callback that is used to define that the option is optional and input value should not be specified. This kind of option is called a flag. opt_required_input_required() function returns a callback that is used to define that the option is required and its value is also required.

Value

Function object

See Also

define_option ParserDef-class defineOptions-package

Examples

```
callback = opt_optional_input_required( input_when_omitted = "70,180" )
callback = opt_optional_input_disallowed( input_when_specified = "TRUE",
                                          input_when_omitted = "FALSE" )
callback = opt_required_input_required()
```

---

define_option *Function to define an option for argument parsing*

Description

define_option function adds a new definition for argument parsing.

Usage

```
## S4 method for signature 'ParserDef,list'
define_option(obj,new_setting)
```

Arguments

obj	ParserDef S4 object
new_setting	list

Details

define_option is a S4 method of ParserDef class. This method adds a definition of argument parsing to a ParserDef object. new_setting argument requires a list that consists of def_name, def_type, long_option, short_option, input_splitter and callback. def_name, def_type, long_option or short_option and callback are required elements. def_name is an identifier of this definition and also works as a name of an element of a list as the final parsing result. def_type is a type to which each input value is cast into. long_option or short_option defines a part of command line options strting from dash such as "–output" and "-o". input_splitter splits input value with the characters specified. Callback is important and defines how the option should be specified. callbacks document describes its detail.

Value

ParserDef object

See Also

ParserDef-class defineOptions-package

Examples

```
parser_def = new_parser_def() |>
    define_option(
        list(
            def_name = "target_range",
            def_type = "integer",
            long_option = "--target-range",
            short_option = "-t",
            input_splitter = ",",
            callback = opt_optional_input_required( input_when_omitted = "70,180" )
        )
    ) |>
    define_option(
        list(
            def_name = "exclude_weekend",
            def_type = "logical",
            long_option = "--exclude-weekend",
            callback = opt_optional_input_disallowed( input_when_specified = "TRUE",
                                                      input_when_omitted = "FALSE" )
        )
    )|>
    define_option(
        list(
            def_name = "output_path",
            def_type = "character",
            long_option = "--output",
            callback = opt_required_input_required()
        )
    )
```

---

new_parser_def *Constructor of ParserDef class*

Description

This is a constructor of ParserDef class.

Usage

```
new_parser_def()
```

Value

ParserDef S4 class object

See Also

`ParserDef-class` `defineOptions-package`

Examples

```
new_parser_def()
```

`ParserDef-class` *ParserDef S4 class*

Description

ParserDef object stores definitions of command line arguments and their parsing.

Details

Package users can create an object of ParserDef class using `new_parser_def` function. `define_option` function adds a new definition for command line parsing. `parse_with_defs` function parses command line arguments based on the definitions of ParserDef object. Each definition searches whether their options are specified or not. Each definition invokes their callbacks and processes specified input, or assign default input values if they are not specified. After callback execution, return value of characters are splitted by input_splitter if input_splitter is specified. Then, the value is cast into def_type. The result values are stored as an element of a list, and each element name is defined by def_name. Remaining arguments are treated as positional arguments.

See Also

`new_parser_def` `define_option` `parse_with_defs` `defineOptions-package`

`parse_with_defs` *Function to parse command line arguments with ParserDef S4 object*

Description

parse_with_defs function parses command line arguments.

Usage

```
## S4 method for signature 'ParserDef,character'
parse_with_defs(obj,cmd_args)
```

Arguments

`obj`	ParserDef S4 object
`cmd_args`	character

Details

parse_with_defs is a S4 method of `ParserDef` class. This method parses command line options with the definitions of ParserDef. It returns a list that holds parsed option values, positional arguments and default values for options not specified.

Value

List (S3 parsed_result class)

`values`	list with values. Each element name is defined by def_name.
`opt_specified`	list with boolean values. Each element name is defined by def_name. Boolean values that represent whether the option are specified in command line arguments or not. FALSE means the value is supplied as a default value through callback mechanism.
`positional`	positional arguments. If there are no positional arguments, NA is assigned.

See Also

`ParserDef-class` `defineOptions-package` `summary.parsed_result`

Examples

```
# In practice, command line arguments can be obtained by commandArgs() function
# with trailingOnly option TRUE.
# command_arguments = commandArgs(trailingOnly = TRUE)

example_string = "input1.txt input2.txt --target-range 60,140 --exclude-weekend --output log.data"
command_arguments = strsplit( example_string, " ")[[1]]

parsed_result = parse_with_defs(parser_def, command_arguments) # parser_def is a ParserDef object
```

`summary.parsed_result` *Summarize parsed_result S3 object*

Description

summary function for parsed_result S3 object.

Usage

```
## S3 method for class 'parsed_result'
summary(object,...)
```

Arguments

`object`	S3 parsed_result class
`...`	Further arguments passed to or from other methods.

Details

summary function for parsed_result S3 object. This enables users to see how values are assigned.

Value

List

`message`	character vector. Description of this list.
`assigned values`	dataframe holding information about definition name(def_name), option names(long_option or short_option), values and how these values are supplied (opt_specified).
`positional arguments`	character vector of positional arguments. If there are no positional arguments, NA is assigned.

See Also

`parse_with_defs`

Examples

```
# In practice, command line arguments can be obtained by commandArgs() function
# with trailingOnly option TRUE.
# command_arguments = commandArgs(trailingOnly = TRUE)

example_string = "input1.txt input2.txt --target-range 60,140 --exclude-weekend --output log.data"
command_arguments = strsplit( example_string, " ")[[1]]

parsed_result = parse_with_defs(parser_def, command_arguments) # parser_def is a ParserDef object
summary(parsed_result)
```

Index