Package: options (via r-universe)

November 16, 2024

Title Simple, Consistent Package Options

Version 0.3.0

Description Simple mechanisms for defining and interpreting package options. Provides helpers for interpreting environment variables, global options, defining default values and more.

License MIT + file LICENSE

URL https://dgkf.github.io/options/, https://codeberg.org/dgkf/options

BugReports https://codeberg.org/dgkf/options/issues

Imports utils

Suggests crayon, knitr, rmarkdown, roxygen2, rcmdcheck, pkgload, withr, testthat (>= 3.0.0)

Config/Needs/website pkgdown

Roxygen list(markdown = TRUE)

VignetteBuilder knitr

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

Config/testthat/edition 3

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

RemoteUrl https://github.com/dgkf/options

RemoteRef HEAD

RemoteSha 266af87306218ca4516bbf0ad8c388e280f0f94d

Contents

`as_params` *Produce* `@param` *roxygen sections for options*

Description

Generate parameter documentation based on option behaviors. Especially useful for ubiquitous function parameters that default to option values.

Usage

```
as_params(...)
```

Arguments

`...`	Character values of options to use. If named arguments are provided, the option description provided as the value is mapped to a parameter of the argument's name.

Value

A character vector of `roxygen2` `@param` tags

See Also

Other options_roxygen2: `as_roxygen_docs()`

Examples

```
options::define_options(
  "whether messages should be written softly, or in all-caps",
  quiet = TRUE
)

#' Hello, World
#'
#' @eval options::as_params("softly" = "quiet")
#'
hello <- function(who, softly = opt("quiet")) {
  say_what <- paste0("Hello, ", who, "!")
  if (quiet) say_what else toupper(say_what)
}
```

as_roxygen_docs *Generate Standalone* ?options *Documentation*

Description

Produce a comprehensive documentation page outlining all your defined options' behaviors.

Usage

```
as_roxygen_docs(
  title = paste(pkgname(env), "Options"),
  desc = default_options_rd_desc(),
  env = parent.frame()
)
```

Arguments

title	An optional, customized title (defaults to "Options")
desc	An optional, customized description of behaviors
env	An environemnt in which to find the associated options object

Value

A character vector of roxygen2 tag segments

See Also

Other options_roxygen2: as_params()

Examples

```
#' @eval options::as_roxygen_docs()
NULL
```

defining_options *Defining Options*

Description

Define options which can be used throughout your package.

Usage

```
define_option(option, ...)

define_options(...)
```

Arguments

option	An option name to use
...	Additional arguments passed to option_spec()

Details

At their simplest, defining options lets you refer to a global option using a shorthand option name throughout your package, with the added benefit of looking for configurations in global options and environment variables.

Value

the package options environment

Functions

- define_option(): Define an option. Unlike define_options(), this function allows detailed customization of all option behaviors. Accepts either an option_spec() object, or an option named followed by arguments to provide to option_spec().
- define_options(): Define multiple options. This function provides a shorthand syntax for succinctly defining many options. Arguments are defined in groups, each starting with an unnamed description argument. For more details see Section *Non-Standard Evaluation*.

Non-Standard Evaluation

define_options() accepts arguments in a *non-standard* way, as groups of arguments which each are used to specify an option (See options_spec()). Groups of arguments must start with an unnamed argument, which provides the description for the argument, followed immediately by a named argument providing the name of option and default value, followed by any additional arguments to provie to options_spec().

The environment in which options are defined is always assumed to be the parent environment. If you'd prefer to specify options in a different environment, this is best done using define_option() or with(<env>, define_options(...)).

Although define_options() provides all the functionality of define_option() in a succinct shorthand, it is only recommended in cases where the overwhelming majority of your options leverage default behaviors. It is encouraged to use define_option() if you repeatedly need more involved definitions to minimize non-standard evaluation bugs.

Examples

```
define_options(
  "Whether execution should emit console output",
  quiet = FALSE,
  "Whether to use detailed console output (showcasing additional
  configuration parameters)",
  verbose = TRUE,
  envvar_fn = envvar_is_true()
)
```

```
define_option(
  "deprecations",
  desc = "Whether deprecation warnings should be suppressed automatically",
  default = FALSE,
  option_name = "MypackageDeprecations",
  envvar_name = "MYPACKAGE_ENVVARS_DEPRECATIONS"
)
```

---

envvar_fns *Generator functions for environment variable processors*

Description

These functions return environment variable processor functions. Arguments to them are used to specify behaviors.

Usage

```
envvar_is(value, ...)

## S3 method for class '`NULL`'
envvar_is(value, case_sensitive = FALSE, ...)

## S3 method for class 'character'
envvar_is(value, case_sensitive = FALSE, ...)

## S3 method for class 'numeric'
envvar_is(value, ...)

## S3 method for class 'logical'
envvar_is(value, case_sensitive = FALSE, ...)

envvar_eval(...)

envvar_eval_or_raw(...)

envvar_is_one_of(values, ...)

envvar_choice_of(values, default = NULL, ...)

envvar_is_true(...)

envvar_is_false(...)

envvar_is_set(...)

envvar_str_split(delim = ";", ...)
```

Arguments

value	A value to test against
...	Other arguments unused
case_sensitive	A logical value indicating whether string comparisons should be case-sensitive.
values	A list or vector of values to match
default	A default value used when a value cannot be coerced from the environment variable value
delim	A character value to use as a delimiter to use when splitting the environment variable value

Value

A function to be used for processing an environment variable value

Functions

- `envvar_is()`: Test for equality with handlers for most atomic R types, performing sensible interpretation of environment variable values.
- `envvar_is(`NULL`)`: environment variable has value `"null"`
- `envvar_is(character)`: environment variable is equal to string `value`
- `envvar_is(numeric)`: environment variable is equal to string representation of numeric `value`
- `envvar_is(logical)`: environment variable is equal to string representation of logical `value`
- `envvar_eval()`: Parse the environment variable value as R code and and evaluate it to produce a return value, emitting an error if the expression fails to parse or evaluate. This option is a sensible default for most R-specific environment variables, but may fail for string literals, and meaningful values that don't conform to R's syntax like `"true"` (see `envvar_is_true()`), `"false"` (see `envvar_is_false()`) or `"null"`.
- `envvar_eval_or_raw()`: Parse the environment variable value as R code and and evaluate it to produce a return value, or falling back to the raw value as a string if an error occurs.
- `envvar_is_one_of()`: For meaningful string comparisons, check whether the environment variable is equal to some meaningful string. Optionally with case-sensitivity.
- `envvar_choice_of()`: Check whether environment variable can be coerced to match one of `values`, returning the value if it matches or `default` otherwise.
- `envvar_is_true()`: Test whether the environment variable is "truthy", that is whether it is case-insensitive `"true"` or `1`
- `envvar_is_false()`: Test whether the environment variable is "falsy", that is whether it is case-insensitive `"false"` or `0`
- `envvar_is_set()`: Test whether the environment variable is set. This is somewhat operating-system dependent, as not all operating systems can distinguish between an empty string as a value and an unset environment variable. For details see `Sys.getenv()`'s Details about its `unset` parameter.
- `envvar_str_split()`: Interpret the environment variable as a delimited list of strings, such as `PATH` variables.

`get_options_env` *Retrieve options environment (experimental)*

Description

The options environment stores metadata regarding the various options defined in the local scope - often the top environment of a package namespace.

Usage

```
get_options_env(env, ...)

## S3 method for class 'options_env'
get_options_env(env, ...)

## S3 method for class 'options_list'
get_options_env(env, ...)

## Default S3 method:
get_options_env(
  env = parent.frame(),
  ...,
  inherits = FALSE,
  ifnotfound = emptyenv()
)
```

Arguments

`env`	An environment in which to search for an options environment
`...`	Additional arguments unused
`inherits`	Whether to search upward through parent environments
`ifnotfound`	A result to return of no options environment is found.

Value

An environment containing option specifications and default values, or `ifnotfound` if no environment is found.

Note

This function's public interface is still under consideration. It is surfaced to provide access to option names, though the exact mechanism of retrieving these names should be considered experimental.

naming *Define Naming Conventions*

Description

Option naming conventions use sensible defaults so that you can get started quickly with minimal configuration.

Usage

```
set_envvar_name_fn(fn, env = parent.frame())

set_option_name_fn(fn, env = parent.frame())
```

Arguments

fn	A callback function which expects two arguments, the package name and option name, and returns a single character value to use as an environment variable name.
env	An environment in which to search for options settings

Value

The callback function fn

Functions

- set_envvar_name_fn(): Set a callback function to use to format environment variable names.
- set_option_name_fn(): Set a callback function to use to format option names.

Defaults

Given a package mypackage and option myoption, the default settings will generate options and environment variables using the convention:

option:

```
mypackage.myoption
```

environment variable:

```
R_MYPACKAGE_MYOPTION
```

This convention is intended to track closely with how options and environment variables are handled frequently in the wild. Perhaps in contrast to the community conventions, an R_ prefix is tacked on to the default environment variables. This prefix helps to differentiate environment variables when similarly named tools exist outside of the R ecosystem.

Setting Alternative Conventions

If you choose to use alternative naming conventions, you must set the callback function *before* defining options. This is best achieved by altering these settings in the file where you define your options.

If you choose to break up your options across multiple files, then it is best to define the collate order for your R scripts to ensure that the options are consistently configured across operating systems.

See Also

naming_formats

Examples

```
set_envvar_name_fn(envvar_name_generic)

set_envvar_name_fn(function(package, name) {
  toupper(paste("ENV", package, name, sep = "_"))
})
```

---

naming_formats *Naming Convention Formatters*

Description

This family of functions is used internally to generate global option and environment variable names from the package name and internal option name.

Usage

```
option_name_default(package, option)  # "package.option"

envvar_name_default(package, option)  # "R_PACKAGE_OPTION"

envvar_name_generic(package, option)  # "PACKAGE_OPTION"
```

Arguments

package, option	The package name and internal option name used for generating a global R option and environment variable name. As these functions are often provided as values, their arguments rarely need to be provided by package authors directly.

Value

A character value to use as the global option name or environment variable name

Functions

- `option_name_default()`: A default naming convention, producing a global R option name from the package name and internal option name (`mypackage.myoption`)
- `envvar_name_default()`: A default naming convention, producing an environment variable name from the package name and internal option name (`R_MYPACKAGE_MYOPTION`)
- `envvar_name_generic()`: A generic naming convention, producing an environment variable name from the package name and internal option name. Useful when a generic convention might be used to share environment variables with other tools of the same name, or when you're confident that your R package will not conflict with other tools. (`MYPACKAGE_MYOPTION`)

See Also

naming

`opt` *Inspecting Option Values*

Description

Inspecting Option Values

Usage

```
opt(x, default, env = parent.frame(), ...)

opt_set(x, value, env = parent.frame(), ...)

opt(x, ...) <- value

opt_source(x, env = parent.frame())

opts(xs = NULL, env = parent.frame())

opt_set_local(
  x,
  value,
  env = parent.frame(),
  ...,
  add = TRUE,
  after = FALSE,
  scope = parent.frame()
)

opts_list(
  ...,
  env = parent.frame(),
```

```
  check_names = c("asis", "warn", "error"),
  opts = list(...)
)
```

Arguments

x, xs	An option name, vector of option names, or a named list of new option values
default	A default value if the option is not set
env	An environment, namespace or package name to pull options from
...	See specific functions to see behavior.
value	A new value to update the associated global option
add, after, scope	Passed to on.exit, with alternative defaults. scope is passed to the on.exit envir parameter to disambiguate it from env.
check_names	(experimental) A behavior used when checking option names against specified options. Expects one of "asis", "warn" or "stop".
opts	A list of values, for use in functions that accept ... arguments. In rare cases where your argument names conflict with other named arguments to these functions, you can specify them directly using this parameter.

Value

For opt() and opts(); the result of the option (or a list of results), either the value from a global option, the result of processing the environment variable or the default value, depending on which of the alternative sources are defined.

For modifying functions (opt_set and opt<-: the value of the option prior to modification

For opt_source; the source that is used for a specific option, one of "option", "envvar" or "default".

Functions

- opt(): Retrieve an option. Additional ... arguments passed to an optional option_fn. See option_spec() for details.
- opt_set(): Set an option's value. Additional ... arguments passed to get_option_spec().
- opt(x, ...) <- value: An alias for opt_set()
- opt_source(): Determine source of option value. Primarily used for diagnosing options behaviors.
- opts(): Retrieve multiple options. When no names are provided, return a list containing all options from a given environment. Accepts a character vector of option names or a named list of new values to modify global option values.
- opt_set_local(): Set an option only in the local frame. Additional ... arguments passed to on.exit().
- opts_list(): Produce a named list of namespaced option values, for use with options() and withr. Additional ... arguments used to provide named option values.

Note

Local options are set with on.exit, which can be prone to error if subsequent calls are not called with add = TRUE (masking existing on.exit callbacks). A more rigorous alternative might make use of `withr::defer`.

```
old <- opt_set("option", value)
withr::defer(opt_set("option", old))
```

If you'd prefer to use this style, see `opts_list()`, which is designed to work nicely with `withr`.

Examples

```
define_options("Whether execution should emit console output", quiet = FALSE)
opt("quiet")

define_options("Whether execution should emit console output", quiet = FALSE)
opt_source("quiet")

Sys.setenv(R_GLOBALENV_QUIET = TRUE)
opt_source("quiet")

options(globalenv.quiet = FALSE)
opt_source("quiet")

define_options("Quietly", quiet = TRUE, "Verbosity", verbose = FALSE)

# retrieve multiple options
opts(c("quiet", "verbose"))

# update multiple options, returns unmodified values
opts(list(quiet = 42, verbose = TRUE))

# next time we check their values we'll see the modified values
opts(c("quiet", "verbose"))

define_options("print quietly", quiet = TRUE)

print.example <- function(x, ...) if (!opt("quiet")) NextMethod()
example <- structure("Hello, World!", class = "example")
print(example)

# using base R options to manage temporary options
orig_opts <- options(opts_list(quiet = FALSE))
print(example)
options(orig_opts)


# using `withr` to manage temporary options
withr::with_options(opts_list(quiet = FALSE), print(example))
```

option_spec *Specify Option*

Description

An option specification outlines the various behaviors of an option. It's default value, related global R option, and related environment variable name, as well as a description. This information defines the operating behavior of the option.

Usage

```
option_spec(
  name,
  default = bquote(),
  desc = NULL,
  option_name = get_option_name_fn(envir),
  envvar_name = get_envvar_name_fn(envir),
  option_fn = function(value, ...) value,
  envvar_fn = envvar_eval_or_raw(),
  quoted = FALSE,
  eager = FALSE,
  envir = parent.frame()
)
```

Arguments

name	A string representing the internal name for the option. This is the short form <option> used within a namespace and relates to, for example, <package>.<option> global R option.
default	Either a quoted expression (if parameter quote == TRUE) or default value for the option. Defaults to an empty expression, indicating that it is unset. The default value is lazily evaluated, evaluated only when the option is first requested unless parameter eager == TRUE.
desc	A written description of the option's effects
option_name, envvar_name	A character value or function. If a character value is provided it is used as the corresponding global option name or environment variable name. If a function is provided it is provided with the package name and internal option name to derive the global option name. For example, provided with package "mypkg" and option "myoption", the function might return global option name "mypkg.myoption" or environment variable name "R_MYPKG_MYOPTION". Defaults to configured default functions which fall back to option_name_default and envvar_name_default, and can be configured using set_option_name_fn and set_envvar_name_fn.
option_fn	A function to use for processing an option value before being returned from the opt accessor functions. For further details see section "Processing Functions".

envvar_fn	A function to use for parsing environment variable values. Defaults to envvar_eval_or_raw(). For further details see section "Processing Functions".
quoted	A logical value indicating whether the default argument should be treated as a quoted expression or as a value.
eager	A logical value indicating whether the default argument should be eagerly evaluated (upon call), or lazily evaluated (upon first use). This distinction will only affect default values that rely on evaluation of an expression, which may produce a different result depending on the context in which it is evaluated.
envir	An environment in which to search for an options envir object. It is rarely necessary to use anything but the default.

Value

An option_spec object, which is a simple S3 class wrapping a list containing these arguments.

Processing Functions

Parameters option_fn and envvar_fn allow for customizing the way values are interpreted and processed before being returned by opt functions.

envvar_fn:

When a value is retrieved from an environment variable, the string value contained in the environment variable is first processed by envvar_fn.

An envvar_fn accepts only a single positional argument, and should have a signature such as:

```
function(value)
```

option_fn:

Regardless of how a value is produced - either retrieved from an environment variable, option, a stored default value or from a default provided to an opt accessor function - it is then further processed by option_fn.

The first argument provided to option_fn will always be the retrieved value. The remaining parameters in the signature should be considered experimental. In addition to the value, the arguments provided to opt(), as well as an additional source parameter from opt_source() may be used.

Stable

```
function(value, ...)
```

Experimental

```
function(value, x, default, env, ..., source)
```

Index