

Package ‘settings’

October 27, 2015

Type Package

Title Software Option Settings Manager for R

Version 0.2.4

Date 2015-10-27

Author Mark van der Loo

Maintainer Mark van der Loo <mark.vanderloo@gmail.com>

Description Provides option settings management that goes beyond R's default 'options' function. With this package, users can define their own option settings manager holding option names, default values and (if so desired) ranges or sets of allowed option values that will be automatically checked. Settings can then be retrieved, altered and reset to defaults with ease. For R programmers and package developers it offers cloning and merging functionality which allows for conveniently defining global and local options, possibly in a multilevel options hierarchy. See the package vignette for some examples concerning functions, S4 classes, and reference classes. There are convenience functions to reset `par()` and `options()` to their 'factory defaults'.

URL <https://github.com/markvanderloo/settings>

BugReports <https://github.com/markvanderloo/settings/issues>

License GPL-3

VignetteBuilder knitr

Imports grDevices, graphics

Suggests testthat, knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2015-10-27 09:51:33

R topics documented:

clone_and_merge	2
defaults	3
inlist	4
is_setting	4
options_manager	5
reset	6
reset_options	7
reset_par	7
settings	8
stop_if_reserved	8

Index	10
--------------	-----------

clone_and_merge	<i>Create a local, altered copy of an options manager</i>
-----------------	---

Description

Local options management.

Usage

```
clone_and_merge(options, ...)
```

Arguments

options	A function as returned by options_manager or <code>clone_and_merge</code> .
...	Options to be merged, in the form of <code>[name]=[value]</code> pairs.

Value

A option manager like options, with possibly different settings.

Details

This function creates a copy of the options manager options, with the same defaults. However, the current settings may be altered by passing extra arguments. Its intended use is to allow for easy merging of local options with global settings in a function call.

Some more examples can be found in the vignette: `vignette('settings', package='options')`.

See Also

[options_manager](#), [reset](#), [defaults](#)

Examples

```
# Create global option manager.
opt <- options_manager(foo=1,bar='a')

# create an altered copy
loc_opt <- clone_and_merge(opt, foo=2)

# this has no effect on the 'global' version
opt()
# but the local version is different
loc_opt()

# we alter the global version and reset the local version
opt(foo=3)
reset(loc_opt)
opt()
loc_opt()

# create an options manager with some option values limited
opt <- options_manager(prob=0.5,y='foo',z=1,
  .allowed=list(
    prob = inrange(min=0,max=1)
    , y    = inlist("foo","bar")
  )
)
# change an option
opt(prob=0.8)
opt("prob")
## Not run:
# this gives an error
opt(prob=2)

## End(Not run)
```

defaults

Request default option values

Description

Request default option values

Usage

```
defaults(options)
```

Arguments

options An option manager, as returned by [options_manager](#) or [clone_and_merge](#)

Value

A list.

See Also

[reset](#)

inlist	<i>Option checkers</i>
--------	------------------------

Description

These functions return a function that is used by the options manager internally to check whether an option set by the user is allowed.

Usage

```
inlist(...)
```

```
inrange(min = -Inf, max = Inf)
```

Arguments

...	comma-separated list of allowed values.
min	minimum value (for numeric options)
max	maximum value (for numeric options)

See Also

[options_manager](#) for examples.

is_setting	<i>Find out if we're setting or getting</i>
------------	---

Description

Utility function checking if we're setting or getting.

Usage

```
is_setting(...)
```

Arguments

...	[key]=[value] pairs of options
-----	--------------------------------

Value

logical, TRUE if ... represents set-options, FALSE if ... represents get-options. An error is thrown if it cannot be determined.

See Also

[stop_if_reserved](#)

options_manager	<i>Create a new options manager.</i>
-----------------	--------------------------------------

Description

Set up a set of options with default values and retrieve a manager for it.

Usage

```
options_manager(..., .allowed)
```

Arguments

... Comma separated [name]=[value] pairs. These will be the names and default values for your options manager.

.allowed list of named functions that check an option (see 'checking options')

Value

A function that can be used as a custom options manager. It takes as arguments a comma separated list of option names (character) to retrieve options or [name]=[value] pairs to set options.

Details

The function options_manager creates an option management function. The returned function can be used to set, get, or reset options. The only restriction of the package is that the following words cannot be used as names for options:

```
.__reset .__defaults
```

For more details and extensive examples see the vignette by copy-pasting this command:

```
vignette("settings", package = "settings")
```

Checking options

Option values can be checked automatically by supplying the options manager with a named list of functions (.allowed) that take an option value and throw an error if it is out-of-range. The functions [inlist](#) and [inrange](#) are convenience functions that create such checking functions for you.

See Also

Reset to default values: [reset](#).
 Retrieve default values: [defaults](#)
 Create a local, possibly altered copy: [clone_and_merge](#)

Examples

```
# create an options register
my_options <- options_manager(foo=1,bar=2,baz='bob')

### Retrieving options
my_options() # retrieve the full options list.
my_options('baz')
my_options('foo')

# When multiple options are retrieved, the result is a list
my_options('foo', 'baz')

### Setting global options
my_options(foo=3,baz='pete')
my_options()
### Reset options to default.
reset(my_options)
my_options()

### Limit the possible values for an option.
my_options <- options_manager( fu="bar",.allowed = list(fu=inlist("foo","bar")) )
```

reset	<i>Reset options to default values</i>
-------	--

Description

Reset options to default values

Usage

```
reset(options)
```

Arguments

options An option manager, as returned by [options_manager](#) or [clone_and_merge](#)

Value

The list of reset options, invisibly.

See Also

[defaults](#)

reset_options	<i>Reset general options in 'options' to factory defaults.</i>
---------------	--

Description

Reset general options in 'options' to factory defaults.

Usage

```
reset_options()
```

See Also

[reset_par](#)

reset_par	<i>Reset graphical options in 'par' to factory defaults.</i>
-----------	--

Description

Reset the [par](#) to R's defaults.

Usage

```
reset_par()
```

Details

Some of `par`'s settings are readonly. These are obviously not reset.

Settings stored in `par` are device-dependent. In practice, most settings in `par` are initially the same across devices. Exceptions we noted are:

- `bg`: background color
- `fin`: figure region dimensions
- `mai`: margin size (inches)
- `pin`: current plot dimensions (inches)
- `plt`: coordinates of the plot region as fractions of the current figure region
- `ps`: point size of text (but not symbols)

Consequently, these options are currently not reset by calling `reset_par()`

See Also

[reset_options](#), [par](#)

 settings

Convenient options settings management for R

Description

Convenient options settings management for R

Details

Reset graphical options of par or options to their factory settings using [reset_par](#) or [reset_par](#).

Create your own option settings manager with [options_manager](#) for projects with many options.

Clone and merge an options manager for easy hierarchical options management using [clone_and_merge](#).

See the vignette for examples, type

```
vignette("settings", package = "settings")
```

at the command-line.

 stop_if_reserved

Check for reserved option names.

Description

Utility function checking for reserved names.

Usage

```
stop_if_reserved(...)
```

Arguments

... Comma-separated [key]=[value] pairs

Value

logical, indicating if any of the keys was reserved (invisibly).

Details

This is a utility function that checks if the keys of the key-value pairs ... contain reserved words.

The reserved words are

._defaults, ._reserved.

If reserved words are encountered in the input an error thrown. The package vignette has examples of its use:

```
vignette('settings', package='options')
```


stop_if_reserved

9

See Also

[is_setting](#)

Index

`clone_and_merge`, 2, 3, 6, 8

`defaults`, 2, 3, 6

`inlist`, 4, 5

`inrange`, 5

`inrange(inlist)`, 4

`is_setting`, 4, 9

`options_manager`, 2–4, 5, 6, 8

`package-settings(settings)`, 8

`par`, 7

`reset`, 2, 4, 6, 6

`reset_options`, 7, 7

`reset_par`, 7, 7, 8

`settings`, 8

`settings-package(settings)`, 8

`stop_if_reserved`, 5, 8