Package `futile.options`

February 19, 2015

Type Package
Title Futile options management
Version 1.0.0
Date 2010-04-05
Author Brian Lee Yung Rowe
Maintainer Brian Lee Yung Rowe <r@nurometic.com>
Depends R (>= 2.8.0)
Description A scoped options management framework
License LGPL-3
LazyLoad yes
Repository CRAN
Date/Publication 2010-04-06 08:57:40

R topics documented:

```
  futile.options-package ....................................................... 1
  OptionsManager .................................................................... 3
```

Index

```
  futile.options-package

  A scoped options management framework

Description

The `futile.options` subsystem provides an easy user-defined options management system that is properly scoped. This means that options created via `futile.options` are fully self-contained and will not collide with options defined in other packages. This package is a self-contained package within the futile suite of libraries.
Details
While R provides a useful mechanism for storing and retrieving options, there is a danger that variable names collide with names defined by a package that a user’s code depends. These types of errors are difficult to detect and should be avoided. Using 'futile.options' addresses this problem by properly scoping variables within its own custom 'namespace'. This is handled by the 'OptionsManager', which acts as a generator for functions that manage user-defined options.

An added benefit to the package is that default values are automatically supported in the creation of the bespoke options manager.

**Author(s)**

Brian Lee Yung Rowe <r@nurometic.com>

**See Also**

OptionsManager

**Examples**

```r
my.options <- OptionsManager('my.options', defaults=list(a=1,b=2))
my.options(a=5, c=3)
my.options('a')
```

**Description**

Included as part of futile is an options subsystem that facilitates the management of options for a particular application. The options.manager function produces a scoped options set within the environment, to protect against collisions with other libraries or applications. The options subsystem also provides default settings that can be restored by calling reset.options.

**Usage**

```r
OptionsManager(option.name, defaults = list())
## Default S3 method:
resetOptions(option.name, ...)
## S3 method for class 'character'
resetOptions(option.name, ...)
```
OptionsManager

```r
## Default S3 method:
updateOptions(option.name, ...)
## S3 method for class 'character'
updateOptions(option.name, key, value, ...) 
options.manager(option.name, defaults = NULL) 
reset.options(option.name, ...) 
```

**Arguments**

- `option.name` The namespace of the options set
- `defaults` A list of default values to use for the new options manager
- `key` A vector of keys in the options that need to be updated
- `value` A vector of values that correspond to the keys above
- `...` Option values to set after resetting

**Details**

Using the options subsystem is simple. The first step is to create a specific options manager for a given namespace by using the `OptionsManager` function. It is possible to specify some default values by passing a list to the default argument. This function returns a specialized function for managing options in the given namespace.

With the new function, options can be set and accessed in an isolated namespace. The options can also be reset using `resetOptions` to the default values.

Note that if multiple values are accessed, to support lists and other complex data structures, the output is a list. If a vector is preferred, pass `simplify=TRUE` as an argument to the user-defined options management function.

Another argument available in the resulting function is `update`, which allows specific values to be updated dynamically rather than via named `key=value` pairs. This is useful in certain situations but can be safely ignored for most situations.

To reset options back to default settings, use the `reset.options` function.

In certain cases, stored options may need to be set programmatically, i.e. their name is constructed dynamically. When this occurs, use `update.options` to set the values.

**Value**

The `OptionsManager` function produces a custom function to manage options for the specified namespace. Use this function to access and set options in your code.

**Author(s)**

Brian Lee Yung Rowe
OptionsManager

Examples

```r
my.options <- OptionsManager('my.options', default=list(a=2,b=3))
my.options(c=4,d='hello')
my.options('b')
my.options('c')

resetOptions(my.options)
my.options('c')

updateOptions(my.options, paste('key',1,sep='.'), 10)
my.options('key.1')
```
Index

*Topic **array**
  - OptionsManager, 3
*Topic **attribute**
  - futile.options-package, 1
*Topic **logic**
  - futile.options-package, 1
*Topic **package**
  - futile.options-package, 1

futile.options
  - (futile.options-package), 1
futile.options-package, 1

options.manager (OptionsManager), 3
OptionsManager, 3, 3

reset.options (OptionsManager), 3
resetOptions (OptionsManager), 3

updateOptions (OptionsManager), 3