Package 'memify'

January 18, 2021

Type Package

Title Constructing Functions That Keep State

Version 0.1.1

Author Bert Gunter

Maintainer Bert Gunter <bgunter.4567@gmail.com>

Description A simple way to construct and maintain functions that keep state i.e. remember their argument lists. This can be useful when one needs to repeatedly invoke the same function with only a small number of argument changes at each invocation.

Depends R (>= 4.0)

Imports utils

License GPL-3

Encoding UTF-8

LazyData true

NeedsCompilation no

Repository CRAN

Date/Publication 2021-01-18 16:40:09 UTC

R topics documented:

`memify-package` *A Simple Framework to Construct and Maintain Functions That Keep State*

Description

This package provides a simple way to construct and maintain versions of R functions that keep their state – i.e. "remember" their explicitly specifed arguments from previous calls. If not overridden, the remembered values of these arguments are automatically reused as the defaults for subsequent calls.

This may be convenient when a function with a large argument list – plotting functions are typical examples – needs to be repeatedly invoked with only a few changes in the arguments at each invocation; or when interacting with a function to determine what parameter values give the most informative results. While there are certainly other ways to do this, the simplicity of this approach may make it a useful alternative.

Details

The package has only one key function: `memify()`. It takes one argument, `f`, a function or a function name as a name/symbol or character string. The result, almost always assigned to a different name, is a new version of `f` that will keep state. It has S3 class `"memified"` that extends the function's original class vector . The original version of `f` of course remains.

Some minor additional convenience functions, `arglist()`, `arglist(x) <-value`, and an update method, `update.memified()`, are also provided. See the examples here and in their repective Help pages for usage.

Note

The *arglist* of a memified function consists only of arguments that were explicitly specified in a prior call. Hence, any formal default arguments that were not changed or specified will not be included in the memified function's arglist, which can be extracted via the `arglist()` function.

Author(s)

Bert Gunter

Maintainer: Bert Gunter <bgunter.4567@gmail.com>

Examples

```
mod <- function(x, b = 5) x %% b
mod.m <- memify(mod) ## or memify("mod")
mod.m(7) ## using default b = 5
mod.m(b = 3) ## using remembered x = 7
mod.m()  ## the same as previous

arglist(mod.m) ## list of all memified arguments
update (mod.m, b = 9) ## silently updates arglist
```

```
arglist(mod.m)

arglist(mod.m) <- list(x=11) ## replaces the arglist with a new list
arglist(mod.m)
## b is no longer memified, so mod's default = 5 is used:
mod.m()

## cleanup
rm(mod.m)
```

`memify` *Enable Functions To Keep State*

Description

Constructs new 'memified' versions of functions that keep state – remember the values of their arguments – between calls.

Usage

```
memify(f, envir = parent.frame())
```

Arguments

`f`	A function (a closure) to convert. If name is a character string then the function with that name is found and used.
`envir`	The `environment` in which the function is defined or found via `get()` when f is a character string.

Details

One should (almost) *never* assign the memified function back to its original function name, as this would replace the original function by its memified version, causing all manner of problems. **You have been warned!**

Value

A function of class `"memified"`, extending the class of f. It is the same as f, except it "remembers" the values of all arguments from its previous calls and uses them as defaults if they are not respecified in the current call. See the examples.

Note

`primitive` functions cannot be memified. They are not closures and some do not make use of named arguments, as they match by position rather than name. However, see the examples below for a workaround of this limitation.

Note also that *unnamed* ... arguments in calls are *not* remembered. They can be included and accessed as usual, but are forgotten when the function returns.

Author(s)

Bert Gunter

See Also

arglist update.memified arglist<-

Examples

```
add2 <- function(a,b) a+b
add2.m <- memify(add2)
add2.m(2,3)  ## a = 2, b= 3, as usual
add2.m(5) ## a =5; b = 3 from previous call
add2.m(b = 10) ## a = 5 from previous call
add2.m() ## both a and b from previous call
z <- 100
add2.m(1,z) ## if not missing, arguments are evaluated as usual
rm(z)

## Also as usual, unexpected arguments produce an error:
## Not run: add2.m(unused = 10)

## Memifying functions with unnamed ... arguments:
sum.m <- memify(function(a,b, ...) sum(a, b, ...))
sum.m(2, 3, 10, 5) ## a =2, b = 3, ... = c(10,5)
sum.m()  ## unnamed arguments are forgotten and not reused!
sum.m( b = 7, 5) ## Is 5 the value for a or ... ?
sum.m()  ## It's for a, following R's standard argument matching rules
arglist(sum.m) ## Is a better way to check argument lists

## memify may be useful in plot functions with many arguments:
plot.m <- memify(plot)
x <- 1:9; y <- runif(9)
plot.m(x,y, col = "blue")
## Change the default type argument and col to "red"
plot.m(col = "red", type = "b")
## make lwd = 2
plot.m(lwd = 2)

## memifying a primitive function:
## exponentiation via '^' is a primitive function that uses positional matching
`^`
## memify a wrapper to convert a primitive to a closure
exp.m <- memify(function(y = 1, x = 0) y^x)
exp.m() ## uses default values
exp.m(2,3) ## y = 2, x = 3
exp.m(x = 5) ## y = 2
exp.m() ## same as previous

## cleanup
rm(add2, add2.m, sum.m, plot.m, exp.m)
```

`memify support functions`

Extract, Update, and Replace Argument Lists of Memified Functions

Description

These functions support the use of memified functions.

Usage

```
arglist(f)

arglist(x) <- value

## S3 method for class 'memified'
update(object, ...)
```

Arguments

`x`, `f`, `object`	A memified function.
`value`	A **named list** of argument values to replace the existing argument list. An attempt will be made to coerce a non-list to a list. If successful, the coerced arglist will be used and a warning reporting the coercion will be issued. Otherwise an error will be thrown.
`...`	Tagged argument-value pairs to add to or replace existing arglist arguments.

Value

`arglist()` returns the existing argument list.

`arglist <-value` replaces the existing argument list with the (possibly coerced) new list.

`update()` (silently) adds additional named arguments to a function's arglist and/or changes the values of any that already exist. `NULL` is invisibly returned.

Warning

The update and replacement functions do not check that valid argument names are used. Invalid arguments can of course cause errors when the memified functions are subsequently invoked.

Note

"arglists", the remembered argument lists of memified functions, consist only of values that have been *explicitly* specified in prior calls of the memified function, or via `update` or `arglist <-value` functionality. Hence default arguments that have not been so changed or specified will not be in the arglists. See `formals` or `args` for ways to extract/change such defaults.

Author(s)

Bert Gunter

See Also

memify

Examples

```
f.m <- memify(function(a = 0, b) sin(a+b))
f.m( b = pi/4) # uses default for a
arglist(f.m)
update(f.m, a= pi/4) ## new default for a
f.m(b = -pi/4)
arglist(f.m) <- list() ## resets arglist
f.m( b = pi/4) ## The original default of a = 0 is used

## cleanup
rm(f.m)
```

Index