Package: envnames (via r-universe)

November 1, 2024

Type Package

Title Track User-Defined Environment Names

Version 0.4.0

Date 2019-01-03

Author Daniel Mastropietro

Maintainer Daniel Mastropietro <mastropi@uwalumni.com>

Description Set of functions to keep track of user-defined environment names (which cannot be retrieved with the built-in function environmentName()). The package also provides functionality to search for objects in environments, deal with function calling chains, and retrieve an object's memory address.

URL https://github.com/mastropi/envnames

BugReports https://github.com/mastropi/envnames/issues

License GPL

Suggests knitr, rmarkdown, testthat

VignetteBuilder knitr

RoxygenNote 6.1.1

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

RemoteUrl https://github.com/mastropi/envnames

RemoteRef HEAD

RemoteSha f9f2e21d94414d588f2605145cc7bef07f9aabed

Contents

envnames-package *Track user-defined environment names*

Description

The main goal of this package is to overcome the limitation of the built-in `environmentName` function of the base package which cannot retrieve the name of an environment unless it is a package or the global environment. This implies that all user-defined environments don't have a "name assigned" that can be retrieved and refer to the environment.

The envnames package solves this problem by creating a lookup table that maps environment names to their memory addresses. Using this lookup table, it is possible to retrieve the name of any environment where an object resides or, from within a function, to retrieve the calling stack showing the function names and their enclosing environment name, i.e. the environment where the functions have been defined. The latter can be done with just a function call which returns a string that can directly be used inside a `cat()` call to display the function name (as opposed to using the R function `sys.call` which does not return a string, but a more complicated object, namely a `call` object from where the string with the function name is still to be extracted to be used inside `cat()`).

Package conventions: all functions in this package follow the underscore-separated and all-lowercase naming convention (e.g. `environment_name()`, `get_obj_address()`, etc.).

Details

The package currently contains 12 visible functions. Following is an overview on how to use the main functions of the package. Please refer to the vignette for further information.

1) Use `get_obj_address` to retrieve the memory address of any object, including environments.

2) Use `environment_name` to retrieve the name of an environment created with `new.env`. The environment can be given as a string containing its 16-digit memory address.

3) Use `obj_find` to find the environments where a given object is defined.

4) Use `get_fun_calling(n)` from within a function to retrieve the name of the calling function `n` levels up in the calling stack together with their enclosing environment name.

5) Use `get_fun_calling_chain` from within a function to get the calling functions stack.

Author(s)

Daniel Mastropietro

Maintainer: Daniel Mastropietro <mastropi@uwalumni.com>

References

Motivation for developing this package:

- A comment by Quantide's instructor Andrea Spano during his "R for developers" course (http://www.quantide.com/courses-overview/r-for-developers) about the impossibility of retrieving the name of user-defined environments.

- A question posted by Gabor Grothendieck at the R-Help forum (https://stat.ethz.ch/pipermail/r-help/2010-July/245646.html)

See Also

environmentName in the base package for the built-in function that retrieves environment names of packages.

exists and find for alternatives of looking for objects in the workspace.

sys.call for other alternatives for retrieving the function calling stack.

Examples

```
library(envnames)
rm(list=ls())

### Example 1: Retrieve the names of user-defined environments (created with new.env())
# Create new environments
env1 <- new.env()                       # Environment in .GlobalEnv
env2 <- new.env()                       # Environment in .GlobalEnv
env3 <- new.env(parent=baseenv())     # Environment whose enclosure or parent environment
                                        # is the base environment
                                        # (as opposed to the global environment)
env_of_envs <- new.env()          # User-defined environment that contains other environments
with(env_of_envs, env11 <- new.env()) # Environment defined inside environment env_of_envs

# Retrieve the environment name
environment_name(env1)               # named array with value "env1" and name "R_GlobalEnv"
environment_name(env3)               # named array with value "env3" and name "R_GlobalEnv"
environment_name(env9)                      # NULL (env9 does not exist)
environment_name(env_of_envs)             # named array with value "env_of_envs" and name
                                            # "R_GlobalEnv"
# (2018/11/19) THE FOLLOWING IS AN IMPORTANT TEST BECAUSE IT TESTS THE CASE WHERE THE ADDRESS-NAME
# LOOKUP TABLE CONTAINS ONLY ONE ROW (namely the row for the env11 environment present in
# env_of_envs), WHICH CANNOT BE TESTED VIA TESTS USING THE testthat PACKAGE BECAUSE IN THAT CONTEXT
# THE LOOKUP TABLE NEVER HAS ONLY ONE ROW!
# (for more info about this limitation see the test commented out at the beginning of
# test-get_env_names.r.
environment_name(env11, envir=env_of_envs)   # "env11"
environment_name(env11)                      # named array with value "env11" and name
                                             # "R_GlobalEnv$env_of_envs"


### Example 2: Retrieve calling functions and their environments
### Note in particular the complicated use of sys.call() to retrieve the call as a string...
# Define two environments
env1 <- new.env()
env2 <- new.env()
# Define function g() in environment env2 to be called by function f() below
# Function g() prints the name of the calling function.
with(env2,
     g <- function(x) {
       # Current function name
       fun_current_name = get_fun_name()

       # Get the name of the calling environment and function
```

```
    fun_calling_name = get_fun_calling()

      # Show calling environment using and not using the envnames package
      cat("Now inside function", fun_current_name, "\n")
      cat("Calling environment name (using environmentName(parent.frame())):  \"",
           environmentName(parent.frame()), "\"\n", sep="")
      cat("Calling environment name (using sys.call(1) inside
          'as.character( as.list(sys.call(1))[[1]]) ))':", " \"",
           as.character( as.list(sys.call(1))[[1]]), "\"\n", sep="")
      cat("Calling environment name (using envnames::get_fun_calling()): \"",
          fun_calling_name, "\"\n", sep="")

      # Start process
      x = x + 2;
      return(invisible(x))
    }
)

# Calling function whose name should be printed when g() is run
with(env1,
     f <- function(x) {
      # Start
      gval = env2$g(x)
      return(invisible(gval))
     }
)

# Run function f to show the difference between using and
# not using the envnames package to retrieve the function calling stack.
env1$f(7)


### Example 3: find the location of an object
# This differs from the R function exists() because it also searches
# in user-defined environments and any environments wihin.
obj_find(f)                   # "env1"
obj_find("f")                 # Same thing: "env1"
obj_find("f", silent=FALSE)   # Same result, but run verbosely

env2$x <- 2
obj_find(x)                   # "env2"

obj_find(nonexistent)         # NULL
```

Index