**The envnames package: an introduction**

If you are used to working with your own environments –for instance when developing a package– you may have been frustated by the output of running a code similar to the following:

myenv <- new.env()

environmentName(myenv)

## [1] ""

The frustration may have come when you see the empty string in the output from the environmentName() function, instead of myenv.

Gladly, the environment\_name() function in the recently released envnames package comes to our rescue in these situations, as explained in the upcoming sections.

**The environment\_name() function does give us the name of the environment**

We can use the environment\_name() function of the package to retrieve the name of the user-defined environment we created above:

library(envnames)

environment\_name(myenv)

## [1] "myenv"

or

library(envnames)

environment\_name(address(myenv))

## [1] "myenv"

where we have used the package’s address() function to show that the **environment name can also be retrieved from the environment’s memory address**, which is where this function becomes truly useful.

In fact, this may come really handy when debugging a program and navigating through environments. In those situations it is not rare to come across a memory address that represents an environment and we may want to know which environment it represents. To this end, we can simply copy & paste the memory address shown in the R console (e.g. "") and run environment\_name("") to get the name of the environment represented by the memory address.

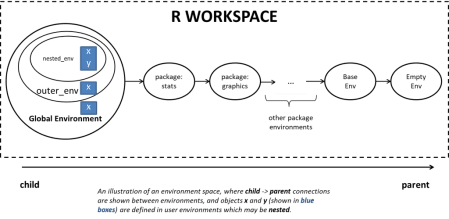
**How does the envnames package work?**

The envnames package is capable of accessing the name of *any* environment –be it a system environment, a package, a namespace, a user-defined environment, or even a function execution environment– by way of a lookup table that maps environment names to their memory addresses. The lookup table is created every time one of the 11 visible functions defined in the package is run, thus updating the map to the latest changes in the workspace.

**Capabilities worth mentioning**

**Looking for an object in nested environments**

The following picture shows an environment space that highlights the connections between package and system environments (child -> parent relationships) and in particular the use of user-defined environments (outer\_env and nested\_env), which are part of the global environment and may be regarded as *nested* environments (within the global environment).



The package includes the obj\_find() function that is able to look for objects in the whole workspace **including user-defined environments**, even if they are **nested in other user environments**.

This is an important feature because the R built-in function exists() does *not* look for objects in user-defined environments, but only in system and package environments. The only way to make exists() look for objects in user-defined environments is to specify them explicitly –*but this doesn’t help if we don’t know where the object may reside!*

The following example illustrates the above limitations of the exists() function, and how such limitations are overcome with the obj\_find() function.

First we define the necessary environments, including objects x and y, as shown in the picture:

outer\_env <- new.env()

outer\_env$nested\_env <- new.env(parent=emptyenv())

x <- 0

with(outer\_env, x <- 3)

with(outer\_env,

{

nested\_env$x <- 5.7;

nested\_env$y <- "phrase"

})

Now we check whether the objects exist using the exists() function:

cat("\n'x' exists?: ", exists("x"), "\n")

cat("\n'y' exists?: ", exists("y"), "\n")

##

## 'x' exists?: TRUE

##

## 'y' exists?: FALSE

We clearly see two important limitations of the exists() function:  
– it does *not* check for existence in *user-defined* environments (y is not found, but it exists in the nested\_env environment),  
– if the object exists, it doesn’t tell us *where* it is found.

Instead, the following calls to the obj\_find() function tell us the whole picture and is informative about the location of the objects (if existing):

cat("\nObject 'x' is found in the following environments:\n",

paste(obj\_find(x), collapse="\n"), "\n", sep="")

cat("\nObject 'y' is found in the following environments:\n",

paste(obj\_find(y), collapse="\n"), "\n", sep="")

cat("\nObject 'nonexisting' is found in the following environments:\n",

paste(obj\_find(nonexisting), collapse="\n"), "\n", sep="")

##

## Object 'x' is found in the following environments:

## outer\_env

## outer\_env$nested\_env

## R\_GlobalEnv

##

## Object 'y' is found in the following environments:

## outer\_env$nested\_env

##

## Object 'nonexisting' is found in the following environments:

Not only is object y found, but object x is found in *all* three environments where it is defined, including the user-defined environments outer\_env and nested\_env, despite the fact that nested\_env is *nested* in outer\_env. The path to reach each object is shown using the $ notation, which is the symbol used to access the object’s value, as achieved by outer\_env$nested\_env$x.

**Looking for an object in a function’s execution environment**

If we are working inside a function, we could also look for objects defined in the function calling chain by specifying include\_functions=TRUE, as shown in the following example:

h <- function() {

x <- 10.37

cat("Object 'x' is found in the following environments:\n",

paste(obj\_find(x, include\_functions=TRUE), collapse="\n"), "\n", sep="")

}

env1 <- new.env()

with(env1,

g <- function() {

x <- 2

h()

}

)

env1$g()

## Object 'x' is found in the following environments:

## env1$g

## eval

## h

## handle

## outer\_env

## outer\_env$nested\_env

## process\_group

## process\_group.block

## R\_GlobalEnv

## timing\_fn

## withVisible

where we see all the (8) function environments where x has been passed during the execution of the code, plus three non-function environments. For now, regular environments cannot be distinguished from function environments in the output returned by obj\_find(), but this will be improved in a future release where the plan is to add the () symbol at the end of function environment names, e.g. env1$g().

**Summary**

We have seen a few ways in which the envnames package can help us work with user-defined environments, namely:  
– use the obj\_find() function to **look for objects** in the workspace, and **retrieve the name of the environments** where they reside, be it a system environment, a package, a namespace, a **user-defined environment** or, when working inside a function, the name of the function whose execution environment is hosting the object.  
– use the environment\_name() function to **find the name of an environment given its memory address** (specially useful in debug contexts)

**Session Info**

This article was generated with [envnames-v0.4.0](https://cran.r-project.org/web/packages/envnames) on the following platform and R version:

## SystemInfo

## sysname Windows

## release 10 x64

## version build 17134

## machine x86-64

## \_

## platform x86\_64-w64-mingw32

## arch x86\_64

## os mingw32

## system x86\_64, mingw32

## status

## major 3

## minor 5.2

## year 2018

## month 12

## day 20

## svn rev 75870

## language R

## version.string R version 3.5.2 (2018-12-20)

## nickname Eggshell Igloo