Want to know an environment's name in R? Use package envnames

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. `"<environment: 0x00000000147499b0>"`) and run `environment_name("<environment: 0x00000000147499b0>")` to get the name of the environment represented by the memory address.

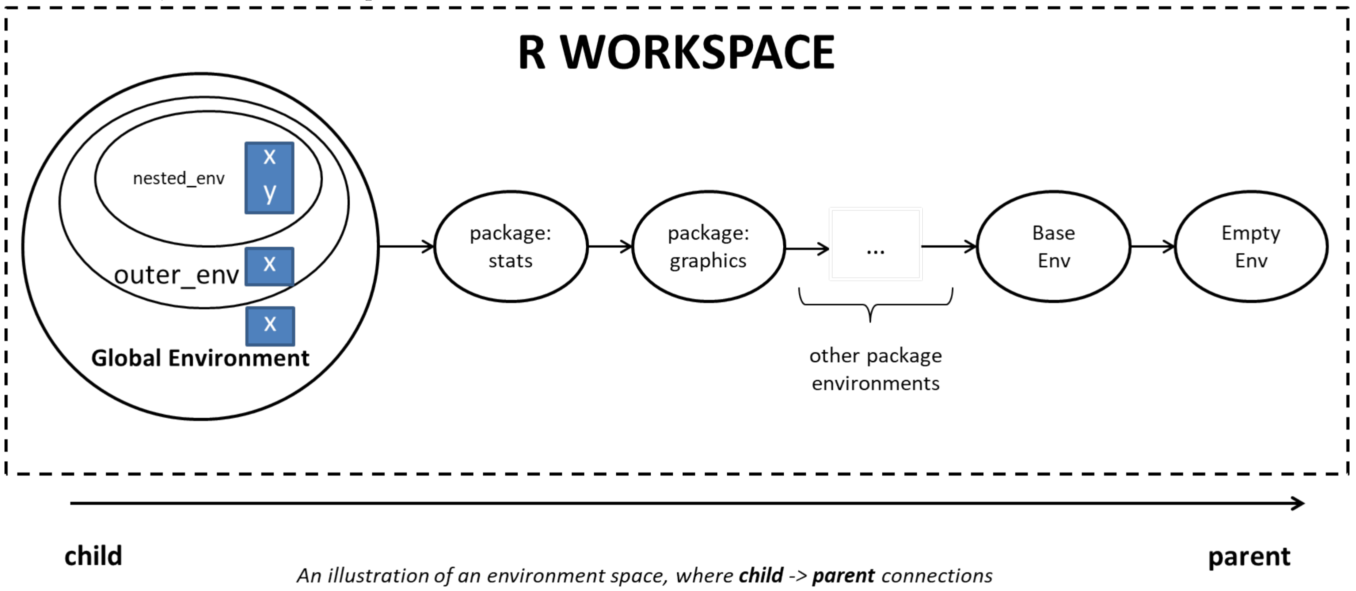
How it works

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 connection between package and system environments (child `->` parent relationships) and in particular the possibility of defining and working with nested environments, which are the topic of this section.



An illustration of an environment space, where ***child*** *->* ***parent*** *connections are shown between environments, and objects* ***x*** *and* ***y*** *(shown in* ***blue boxes****) are defined in user environments which may be* ***nested****.*

The package includes the obj_find() function that is able to look for objects in the whole workspace **including nested environments**. The following example, where environments and objects are defined to mimic the situation shown in the above picture, illustrates the search for objects x and y in the whole workspace.

First we define objects x and y:

```
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 look for the objects using obj_find(x) and obj_find(y):

```
cat("\nObject 'x' is found in the following environments:\n",
    paste(obj_find(x), collapse="\n"), "\n", sep="")
##
## Object 'x' is found in the following environments:
## outer_env
## outer_env$nested_env
## R_GlobalEnv
cat("\nObject 'y' is found in the following environments:\n",
    paste(obj_find(y), collapse="\n"), "\n", sep="")
##
## Object 'y' is found in the following environments:
## outer_env$nested_env
```

We see that object x is found in all three environments where it is defined, including the nested_env environment that is *nested* in outer_env. The path to reach each object is shown using the $ notation. Note also that the global environment shows up with the name R_GlobalEnv.

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 3 regular 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:
- **look for objects** in the workspace, including *nested environments*, and *function execution environments*.
- **see the name of the environments** where those objects 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.
- **know the name of an environment given its memory address** (specially useful in debug contexts)

To learn more about further capabilities provided by the package, I invite you to take a look at the **vignette**.

Finally, if you decide to install the package and use it, I would be very happy to learn about your use cases, so just drop me a comment below.

References

Motivation for writing this package: https://stat.ethz.ch/pipermail/r-help/2010-July/245646.html (question by Gabor Grothendieck at a forum on R in 2010)

Acknowledgements

I would like to acknowledge Andrea Spanò's contribution in inspiring the development of this package during the R for Developers course he gave at Quantide.

Session Info

This article was generated with envnames-v0.4.0 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
```