**Introduction**

This week I was working on a package which included a function factory. A function factory is a function which returns a function. The problem I faced was that when I was running R CMD check on my package, the check informed me my package had several issues which on first glance were confusing and seemingly shouldn’t have been reported. In this blog post, we’ll discover why.

**Function Factory Example**

Let’s write a function factory. We will use a function factory called power1() to create two additional functions square() and cube().

power1 <- function(exp) {

function(x) {

x ^ exp

}

}

square <- power1(2)

cube <- power1(3)

By assigning power1(2) to the object square, we have created a new function, square(), which when given a value for x will return the result of x to the power of two. Similarly the cube() function will return the result of x to the power of three. Let’s see this in action.

square(3)

# [1] 9

cube(3)

# [1] 27

**Could Not Find Function “power1”**

I was working on a package which contained my function factory in file A and multiple “child” functions, which were generated by that function, in file B. Everything seemed to be working fine; locally I could install the package and the R CMD check ran without any ERRORs, WARNINGs or NOTEs. However when I ran the code within my CI/CD pipeline, the R CMD check gave me the following set of messages.

R CMD check results

1 error | 3 warnings | 2 notes

checking for missing documentation entries ... ERROR

Error: cannot source package code:

could not find function "power1"

Execution halted

checking S3 generic/method consistency ... WARNING

Error: cannot source package code:

could not find function "power1"

Execution halted

See section 'Generic functions and methods' in the 'Writing R

Extensions' manual.

checking replacement functions ... WARNING

Error: cannot source package code:

could not find function "power1"

Execution halted

The argument of a replacement function which corresponds to the right

hand side must be named 'value'.

checking for code/documentation mismatches ... WARNING

Error: cannot source package code:

could not find function "power1"

Execution halted

checking R code for possible problems ... NOTE

Error: cannot source package code:

could not find function "power1"

Execution halted

checking Rd \usage sections ... NOTE

Error: cannot source package code:

could not find function "power1"

Execution halted

The \usage entries for S3 methods should use the \method markup and not

their full name.

See chapter 'Writing R documentation files' in the 'Writing R

Extensions' manual.

This had me really stumped for a while – especially as the messages given by R CMD check were talking about things that were unrelated to my code as I have no S3 methods in my package, my documentation was up to date and the function was definitely there and checked into my git repository. So why then was this not working within my CI/CD pipeline? Finally the penny dropped and I finally realised that it is because when R is checking the package, it must be sourcing square() and cube() *before* it has sourced power1(). Therefore it cannot assign the output of power1() to square and cube since it doesn’t “exist” at that point.

**The Solution**

The solution, it turns out, is very straight forward. We must tell R to collate the files in a particular order and we can do this by specifying the order in the Collate: field of the DESCRIPTION file. But even better, as I am using roxygen2, I can use the @include tag to state that one file needs another to work. Therefore as I need to make sure that file power1.R is loaded before square.R and cube.R, I simply include @include power1.R in the other files. roxygen2 takes care of ordering the Collate: field to satisfy these restrictions. As another handy tip, let’s say square() and cube() were defined in a file together which is separate from the power1.R file, we can include the following lines of code at the top of the child function file.

#' @include power1.R`

NULL

And roxygen2 will again take care of the Collate: field for us automatically.

As for why everything worked locally but not in my CI/CD pipeline, I can only assume that as the CI/CD pipeline runs on a different machine, R is using some different method to source and collate the files. So it is better to be safe than sorry and explicitly include the order of the files in the Collate: field.

**Conclusion**

To conclude, if you plan on including a function factory in your package where the factories’ child functions are in a different file, it is really important that you tell R the order in which it should source these files. You can achieve this simply with roxygen2 by using the @include tag – specifying the function factory file name – within the file(s) of the function(s) generated by the function factory. roxygen2 will then automatically fill and sort the Collate: field within your DESCRIPTION file.