# Debugging R code using R, RStudio and wrapper functions

*Han Oostdijk (original John Mount from Win-Vector LLC)*

*11 april 2016*

## R libraries used

none (base R only)

## Introduction

This document is an extract of the YouTube video and accompanying github code by John Mount from Win-Vector LLC applied on a very simple R function. It shows how a failing R function can be debugged more easily by saving in an external file all arguments and the name of the function. This is especially helpful if this function is called not on the global level but on a (deeply) nested level when it could be difficult to determine with what arguments the function is called when it fails.

What is different from John's original work:

- the two wrapper functions are combined in one
- the error message is improved (in my opinion)
- introduction of the **trace** function to facilitate the debugging process.

## How to use this debugging functionality

It can be used in the following way:

- let us say that **xxx** is the failing function that we want to debug
- replace all calls to **xxx** by a call to the wrapper function. See Wrapper functions.
- after running the changed code an RDS file is created with the arguments of the **xxx** function when that failed.
- load the contents of the RDS file in e.g. variable *problem*. See Load arguments.
- use the **trace** function to activate debugging of the **xxx** function. See Activate Debugging
- call **xxx** with the saved arguments in *problem*. Due to the **trace** function it will be executed in debugging mode.
- use the **untrace** function to deactivate debugging of the **xxx** function. See Deactivate debugging

NB. this method of debugging will only work when the failing function has no external dependencies; i.e. when the working of the function is fully determined by its arguments. It will/could not work when e.g. global variables are used, when there is a dependency on *options* or when the arguments can't be properly saved (e.g. the case when an argument is an XML document in useInternalNodes format).

## Wrapper functions

John created two wrapper functions **DebugFn** and **DebugPrintFn** that differ only in the fact that in the last one the result of the function is also printed. According to John it often occurs in graphical software that expressions are only evaluated at the moment when they have to be printed. Because these wrapper functions are so similar I have combined the two in the function **DebugPrintynFn**. It is called with the following arguments

- the first argument is the name of the RDS file that will contain a list with the arguments when/where the funtions fails and the name of the function.
- the second argument is a boolean indicating if a *print* statement is to be inserted
- the third argument is a character string with the name of the function (**xxx** in our case)
- the other arguments are the arguments for the **xxx** function.

```
DebugPrintynFn <- function(saveFile, prnt = F, fn, ...) {
  args <- list(...)
  tryCatch({
    res = do.call(fn, args)
    if (prnt == T) {
      print(res)
    }
    res
  },
  error = function(e) {
    saveRDS(object = list(fn = fn, args = args), file = saveFile)
    em = paste0(unlist(strsplit(as.character(e),':')),collapse ='\n')
    em = substring(em,1,nchar(em)-1)
    e$message <- paste0("\nWrote '", saveFile, "' on catching:\n'", em, "'")
    stop(e)
  })
}
```

## Test function xxx

We use the following simple function **xxx** that gives an error when called with *force_error* = *F* (because the function xprintf is not defined here).

```
xxx <- function(v1, v2, force_error = F)
{
  v0 = 'first'
  if (force_error == F) {
    sprintf('%s - %s - %s', v0, v1, v2) # okay
  } else {
    xprintf('%s - %s - %s', v0, v1, v2) # error
  }
}
```

## Call xxx via the wrapper function

We call **xxx** twice; once with and once without forcing the error.

```
v1 = 'John' ; v2 = 'Mount'
resokay  = tryCatch(
  DebugPrintynFn('problem.RDS',prnt=T,'xxx',v1,v2,F),
  error = function(e) { print(e) })
```

```
## [1] "first - John - Mount"
```

```
resfault = tryCatch(
  DebugPrintynFn('problem.RDS',prnt=T,'xxx',v1,v2,T),
  error = function(e) { print(e) })
```

```
## <simpleError in xxx("John", "Mount", TRUE):
## Wrote 'problem.RDS' on catching:
## 'Error in xxx("John", "Mount", TRUE)
##  could not find function "xprintf"'>
```

## Load arguments from *problem.RDS*

Because the last call to **xxx** caused an error the wrapper function created the RDS file with the arguments as shown here:

```
problem <- readRDS('problem.RDS')
print(problem)
```

```
## $fn
## [1] "xxx"
##
## $args
## $args[[1]]
## [1] "John"
##
## $args[[2]]
## [1] "Mount"
##
## $args[[3]]
## [1] TRUE
```

## Activate debugging

Activate the trace and display the function to see if this has been done. We use the construction *problem$fn* to generalize but in this case we know that the function is **xxx** so we could have typed this directly.

```
trace(problem$fn,browser)
```

```
## [1] "xxx"
```

```
body(problem$fn)
```

```
## {
##     .doTrace(browser(), "on entry")
##     {
##         v0 = "first"
##         if (force_error == F) {
##             sprintf("%s - %s - %s", v0, v1, v2)
##         }
##         else {
##             xprintf("%s - %s - %s", v0, v1, v2)
##         }
##     }
## }
```

```
args(problem$fn)
```

```
## function (v1, v2, force_error = F)
## NULL
```

## Call xxx with the failing arguments

We will not execute the following because we know it will fail and we can't debug in this document. But in practice we would execute this. Because of the trace we would enter debugging mode with the same arguments as when it failed before.

```
do.call(problem$fn,problem$args)
```

## Deactivate debugging

Remove the tracing code from the **xxx** function and display the function to see if this has been done.

```
untrace(problem$fn)
body(problem$fn)
```

```
## {
##     v0 = "first"
##     if (force_error == F) {
##         sprintf("%s - %s - %s", v0, v1, v2)
##     }
##     else {
##         xprintf("%s - %s - %s", v0, v1, v2)
##     }
## }
```

```
args(problem$fn)
```

```
## function (v1, v2, force_error = F)
## NULL
```

## Session Info

```
sessionInfo()
```

```
## R version 3.2.4 (2016-03-10)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 10586)
##
## locale:
## [1] LC_COLLATE=English_United States.1252  LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252 LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] knitr_1.12.3
##
## loaded via a namespace (and not attached):
##  [1] magrittr_1.5    formatR_1.3     tools_3.2.4     htmltools_0.3   yaml_2.1.13
##  [6] stringi_1.0-1   rmarkdown_0.9.5 stringr_1.0.0   digest_0.6.9    evaluate_0.8.3
```