**Introduction**

When speed and memory efficiency is important, the data.table package is one of the ways to improve those aspects of our R code dramatically. Including data.table in a package also comes with the added benefit of only importing the methods package, which is part of base R. We must also however pay attention to correctly importing and using methods, as data.table handles data.frame subsetting operators in a special way. This post is mostly a lesson learned for future self on how I did not pay attention and what I found out investigating.

**TL;DR if you just want something useful**

* Use options(datatable.verbose = TRUE) to see useful logging information
* If you are getting weird errors with subset methods, check if data frame methods do not get called instead of the data table ones (e.g. running traceback() after the error occurs)
* If so, check if data.table:::cedta() returns FALSE for your package. And if it does, check if you import data.table in the NAMESPACE file of your package

**A somewhat reproducible example of the issue**

Imagine a very simple function that takes a data table and sums a column with a name provided via the y argument, grouped by the column name provided via the by argument. An oversimplified definition and example use with the mtcars dataset could look as follows:

sumData <- function(dt, y, by) dt[, sum(get(y)), by = by]

mtcarsdt <- data.table::as.data.table(datasets::mtcars)

sumData(mtcarsdt, "disp", "gear")

## gear V1

## 1: 4 1476.2

## 2: 3 4894.5

## 3: 5 1012.4

So far so good, everything works great. Now we put our awesome function into a nice package called dtexample. Add some roxygen documentation, add data.table into Imports in our DESCRIPTION, try to install our package. All still works. Run R CMD check for good measure and get 0 errors, 0 warnings and 0 notes, like a boss!

Now let’s see our function in action, from within the new package:

dtexample::sumData(mtcarsdt, "disp", "gear")

Error in get(y) : object 'disp' not found

Oops. Something went wrong. Debugging such an issue can be tricky, especially if this happened in a more realistic setting, such as writing the function across multiple days and having a more complicated function than a one-liner. Most often the issue is inside the actual code, especially when passing around more complicated quoted expressions into data table’s subsetting machinery.

**Traceback and datatable.verbose to the rescue**

Let us look at the traceback() to get some insight into what is going on:

traceback()

## 5: get(y)

## 4: `[.data.frame`(x, i, j)

## 3: `[.data.table`(dt, , sum(get(y)), by = by) at sumData.R#12

## 2: dt[, sum(get(y)), by = by] at sumData.R#12

## 1: dtexample::sumData(dt, "disp", "gear")

Note the 4: despite the object being a data table (which is also confirmed by the third line of the traceback), the data frame method was called. It would also seem that this was deliberate on data table’s side. Let us turn on the datatable.verbose option and see what it has to say:

options(datatable.verbose = TRUE)

dtexample::sumData(mtcarsdt, "disp", "gear")

## cedta decided 'dtexample' wasn't data.table aware. Here is call stack with [[1L]] applied:

## [[1]]

## dtexample::sumData

##

## [[2]]

## `[`

##

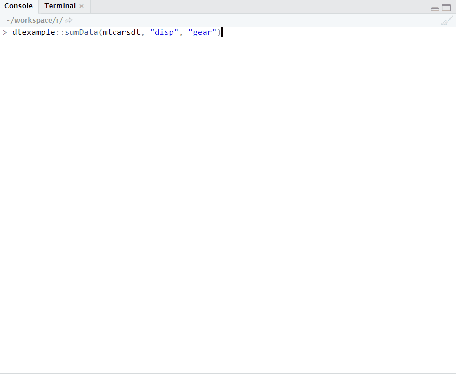
## [[3]]

## `[.data.table`

##

## [[4]]

## cedta



Traceback and cedta()

**So what is this cedta()?**

Looking at data table’s verbose output, we immediately notice this message:

cedta decided ‘dtexample’ wasn’t data.table aware. Here is call stack with [[1L]] applied:

So, what is this cedta() and why is it making such decisions? Let us look how we get from subsetting a data table to a function deciding that our package is not data table aware. Examining the first rows of the body of data.table:::[.data.table we can see that the subset method first examines the output of cedta() and if its results is FALSE, calls the data frame methods. This answers our question of why a data frame method was called:

if (!cedta()) {

Nargs = nargs() - (!missing(drop))

ans = if (Nargs < 3L) {

`[.data.frame`(x, i)

}

else if (missing(drop))

`[.data.frame`(x, i, j)

else `[.data.frame`(x, i, j, drop)

if (!missing(i) & is.data.table(ans))

setkey(ans, NULL)

return(ans)

}

Now looking into data.table:::cedta() itself we see that in case topenv(parent.frame(n)) is not a namespace, cedta() happily returns TRUE. This explains why our function worked when it was defined and run from the global environment. However, in case we are in the context of a namespace, our namespace must satisfy at least one of eight conditions:

ans = nsname == "data.table" ||

"data.table" %chin% names(getNamespaceImports(ns)) ||

(nsname == "utils" && exists(

"debugger.look",

parent.frame(n + 1L)

)) ||

(nsname == "base" && all(c("FUN", "X") %chin% ls(parent.frame(n)))) ||

(nsname %chin% cedta.pkgEvalsUserCode && any(

sapply(sys.calls(), function(x)

is.name(x[[1L]]) && (x[[1L]] == "eval" || x[[1L]] == "evalq"))

)

) ||

nsname %chin% cedta.override ||

isTRUE(ns$.datatable.aware) ||

tryCatch(

"data.table" %chin% get(

".Depends",

paste("package", nsname, sep = ":"),

inherits = FALSE

), error = function(e) FALSE

)

Out of which the most relevant for us is:

"data.table" %chin% names(getNamespaceImports(ns))

When I first saw this, I was like (probably more than 50% of the sentence self-censored):

No way. I could not possibly be so stupid to forget to import data table in the NAMESPACE! *(… of course I could)*

So, about a minute later, place @import data.table into the roxygen tags, regenerate the NAMESPACE, re-install the package and all works great.

**How could I possibly fail to import anything from data.table and find out earlier?**

I think the reason (apart from plain forgetting the obvious) is a combination of the following:

* the subsetting operator is such second nature, that it just did not occur to me to import it with the @importFrom tag and I rarely use @import on entire packages
* R CMD check was successful with no notes, warning or errors, again because even if I usually relatively strictly use qualified calls, the subsetting would seem very unnatural like that. There was therefore no mention of data.table:: in the entire code and the checking procedure had nothing to complain about
* the data table method actually did dispatch correctly, so only after a closer look we see the data frame method kicking in. The first thing to investigate (most of the time correctly) is the actual implementation of what is going on with the expressions inside the subsetting operator, especially when passing around and evaluating quoted expressions

So, if you ever see cedta() making decisions about data table awareness, check your NAMESPACE. Maybe you have just missed the obvious as I did. Happy data tabling!