tidyverse / rlang

Documentation question: how does one convert a string to a quosure? #116

Closed **JohnMount** opened this issue 4 days ago · 9 comments

JohnMount commented 4 days ago • edited

I have skimmed the dplyr/tidyeval/rlang documentation and tutorials and I don't remember or see how to convert a string to a quosure easily. What I want to do (and I think it is an important use case) is take the name of a column as a string from some external source (say from colnames(), or from the yarn-control block of an R-markdown document) and then use that string as a variable name. It looks like to do that you have to promote the string up to a quosure- and that is the part I don't know how to do in pure tidyeval idiom. I've tried things like `quo()`, but I am missing something.

Below is a specific example with a work-around that shows the effect I want. The only question is how does one produce the variable varQ from the value stored in varName (again, assuming the value stored is a string and not known to the programmer)?

```
# devtools::install_github("tidyverse/dplyr")
library("dplyr")



##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union


packageVersion("dplyr")


## [1] '0.5.0.9004'


library("wrapr")

# imagine this string comes from somewhere else
varName <- 'disp'



# The following does not currently work as
# rlang/tidyeval/dplyr is expecting a
```

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Notifications

Unsubscribe

You're receiving notifications because you commented.

4 participants

```
# quousure to represent the variable name,
# not a string.
mtcars %>%
  select(!!varName)



## Error: `"disp"` must resolve to integer column positions, not string



# How does one idiomatically
# create a quosure varQ that refers
# to the name stored in varName?
# Here is a work-around using wrapr
# to show the desired effect:
stringToQuoser <- function(varName) {
  wrapr::let(c(VARNAME = varName), quo(VARNAME))
}
varQ <- stringToQuoser(varName)

# the code we want to run:
mtcars %>%
  select(!!varQ) %>%
  head()



##                   disp
## Mazda RX4          160
## Mazda RX4 Wag      160
## Datsun 710         108
## Hornet 4 Drive     258
## Hornet Sportabout  360
## Valiant            225
```

lionel- commented 4 days ago — Member

you can use `sym()` or `syms()`. You can unquote these symbols directly in capturing functions:

```
select(df, !! sym("foo"))
select(df, !!! syms(c("foo", "bar")))
```

or you can unquote them while creating quosures (which is actually the same mechanism):

```
quo(!! sym("foo"))
quo(list(!!! syms(letters)))
```

 **lionel-** closed this 4 days ago

JohnMount commented 4 days ago

Thanks, just a note: that doesn't work until you explicitly import rlang (dplyr seems to not share it).

```
library("dplyr") # installed today
packageVersion("dplyr")
# [1] ‘0.5.0.9004’
select(mtcars,  !! sym("disp"))
# Error in (function (x)  : could not find function "sym"
```

```
library('rlang')
packageVersion("rlang")
# [1] '0.0.0.9018'
select(mtcars,  !! sym("disp"))
```

lionel- commented 4 days ago Member

You can qualify: `rlang::sym()`

JohnMount commented 4 days ago • edited

Also I don't think `varQ = quo(sym(varName))` works (and it is the bit I really need).

```
library("dplyr")
library("rlang")
varName <- 'disp'
varQ = quo(sym(varName))
varQ
# <quosure: global>
# ~sym(varName)
select(mtcars, varQ)
#  Error: `varQ` must resolve to integer column positions, not formula
```

Even if that did work, it would be dangerous as it looks like it is hanging on a reference to `varName` (instead of taking the value at the current moment), meaning if we later changed the value of `varQ` the select could (do to lazy eval) become a different result (I call this a dragging reference). This drives bugs sort of like this one: tidyverse/dplyr#2455 .

Can you please re-open this issue until we find a working pure `rlang` solution?

hadley commented 4 days ago Owner

But you don't need to create a quosure here; just pass in a symbol (which you can also create with `as.name()` if you don't want to import rlang)

👍 1

lionel- commented 4 days ago Member

When you're programming with NSE functions, you are building an expression. Unquoting makes it possible to change parts of the expression and is what a programmer should focus on:

```
varName <- "disp"
varQ <- quo(!! sym(varName))  # unquoting the symbol
select(mtcars, !! varQ)  # unquoting the quosure
```

Note that the quosure is superfluous here as Hadley mentions, since you're not referring to symbols from the contextual environment.

> it would be dangerous as it looks like it is hanging on a reference to varName

I don't understand what you mean. In the snippet above we build a quosure containing the value of `sym(varName)` which is the symbol `disp` . Note that you can also unquote literal values (i.e. vectors)

just like you can unquote expressions (though in this case `select_vars()` expects column positions rather than actual columns; it evaluates expressions in an environment where column symbols evaluate to column positions).

👍 1

MZLABS commented 4 days ago • edited

John Mount here again (different account, sorry).

I see the later solutions do work, and than you for that. I also understand answering questions is a volunteer activity, so I appreciate you working on this for me.

I had a typo in my attempt to use the answer, and I apologize for that. But variations on that idea do not work even if I type it "correctly":

```
library("dplyr")
library("rlang")
varName <- 'disp'
varQ = quo(sym(varName))
select(mtcars, !! varQ)
# Error: `sym(varName)` must resolve to integer column positions, not symbol
```

To answer some expressed concerns and set some context I have some follow-up, but all my questions are now answered.

As far as the appearance of unbound variables and CRAN check. If I had been submitting the above code to CRAN I would have added the following line above the let-block:

```
VARNAME <- NULL # mark variable symbol as not an unbound reference
```

The reason I asked for "quosure" is: I thought that was all `rlang` accepted. I had tested it does not take strings and base-R formulas, so I was just asking for what I thought was the help I needed based on my state of knowledge.

I do not mind importing `rlang` or qualifying `rlang::sym`, I only mentioned that `dplyr` did not re-export `rlang::sym` to point out the difficulty in discovering that command starting from a `dplyr` task (i.e. working on a `select()`).

The latest solution does work (and thank you for it):

```
library("dplyr")
library("rlang")
varName <- 'disp'
varQ = quo(!! sym(varName))
mtcars %>%
  select(!! varQ) %>%
  head()
#                   disp
#Mazda RX4           160
#Mazda RX4 Wag       160
#Datsun 710          108
#Hornet 4 Drive      258
#Hornet Sportabout   360
#Valiant             225
```

It also appears to not have the captured reference name issue. Since the captured reference issue is not in this variation (there is no visible reference to the variable name "varName" in "varQ") there isn't

much point going more into it. But the rough idea is: if a `quosure` worked by capturing `varName` and `varName` changed between when we thought we set it and when we used the value we would not want the new value to enter into the calculation. The effect (albeit in another context) was discussed at length in my linked issue tidyverse/dplyr#2455 which was supposed to be a worked example by analogy. But as I said, I don't think the current solutions hold a reference to the `varName` (they seem to properly go after the value) so we don't have to worry if we have this issue or not. Typically R doesn't have these issues due to its copy by value semantics, but anywhere names are directly used I worry about possible (unintended, and therefore undesirable) reference-like semantics leaking in.

Also I understand `sym()` this is not the form you would prefer and that `as.name()` can be used.

```
library("dplyr")
varName <- 'disp'
varQ = as.name(varName)
mtcars %>%
  select(!! varQ) %>%
  head()
#                   disp
#Mazda RX4           160
#Mazda RX4 Wag       160
#Datsun 710          108
#Hornet 4 Drive      258
#Hornet Sportabout   360
#Valiant             225
```

Finally I did work on this before asking. I tried all of:

```
varQ = as.formula(~varName)
varQ = quo(!! varName)
varQ = quote(varName)
varQ = quote(!! varName)
```

And none of the above worked.

Frankly I was guessing, but the reason I was guessing is I did not find a worked example of converting a string to something `rlang` is willing to use as variable name. That is: guessing was not my first choice.

I had tried `help(quo)`, `help(UQ)` (`!!` 's equiv) and neither of them mentions, links to, or has an example of `sym()` or `as.name()`.

Anyway thank you very much for your solutions.

hadley commented 4 days ago Owner +😀

You can also use the `.data` pronoun to avoid both R CMD check notes and the need to convert to symbols:

```
varName <- "disp"
mtcars %>% select(.data["disp"])
```

(Well, you will be able to once tidyverse/dplyr#2718 is merged)

lionel- commented 4 days ago • edited Member +😀

> I worry about possible (unintended, and therefore undesirable) reference-like semantics leaking in.

It's not about reference semantics, it's about delayed evaluation. We're building an expression, sometimes in several steps, and if the value of some symbols changes before evaluation actually happens this could be a problem. To work around this, you can unquote values rather than symbols, or you could make sure the symbols are in read-only environments (e.g. by building an appropriate quosure).

> I had tested it does not take strings and base-R formulas

tidyeval works with pure expressions. The only adjustment we make is that quosures self-evaluate within their environments (with overscoped data attached). This is why the following expressions are completely equivalent:

```
select(mtcars, "cyl")

var <- "cyl"
select(mtcars, !! var)
```

This doesn't work because `select()` doesn't work with strings but with column positions (or with expressions evaluating to column positions). Hence the following works:

```
select(mtcars, cyl)

var <- sym("cyl")
select(mtcars, !! var)
```

Alternatively, you can also supply the values it understands (column positions):

```
select(mtcars, 1)

var <- 1
select(mtcars, !! var)
```

Write Preview

Leave a comment

Attach files by dragging & dropping or selecting them.

Styling with Markdown is supported

Comment