In August of 2003 Thomas Lumley added bquote() to [R](https://www.r-project.org/) 1.8.1. This gave R and R users an explicit Lisp-style quasiquotation capability. bquote() and quasiquotation are actually quite powerful. Professor Thomas Lumley should get, and should continue to receive, a lot of credit and thanks for introducing the concept into R.

In fact bquote() is already powerful enough to build a version of dplyr 0.5.0 with quasiquotation semantics quite close (from a user perspective) to what is now claimed in tidyeval/rlang.

Let’s take a look at that.

For our example problem: suppose we wanted to average a ratio of columns by groups, but want to write code so that what columns are to be used is specified elsewhere (perhaps coming in as function arguments). You would write code such as the following in dplyr 0.7.\*:

suppressPackageStartupMessages(library("dplyr"))

# define our parameters

# pretend these come from far away

# or as function arguments.

group\_nm <- as.name("am")

num\_nm <- as.name("hp")

den\_nm <- as.name("cyl")

derived\_nm <- as.name(paste0(num\_nm, "\_per\_", den\_nm))

mean\_nm <- as.name(paste0("mean\_", derived\_nm))

count\_nm <- as.name("group\_count")

And then you would execute something like the following. We are not running the following block (so there is no result), as we have dplyr 0.5.0 installed, which does not yet accept the quasiquotation notation.

# apply a parameterized pipeline using rlang::!!

mtcars %>%

group\_by(!!group\_nm) %>%

mutate(!!derived\_nm := !!num\_nm/!!den\_nm) %>%

summarize(

!!mean\_nm := mean(!!derived\_nm),

!!count\_nm := n()

) %>%

ungroup() %>%

arrange(!!group\_nm)

Writing the above code requires two ideas:

* quasiqotation (marking some symbols for substitution, in this case using !! as the marker). As we said, quasiquotation has been available in R since 2003.
* Using := as a substitute for = to allow substitution marks on the left-hand sides of assignment like expressions without triggering syntax errors. := is an unused left-over assignment operator that has low-precedence (like assignments do), but not all the syntactic restrictions of =. This is an idea used in data.table for quite some time, and is actually [first visible in dplyr when used as part of dplyr‘s data.table adapter in May 2013](https://github.com/tidyverse/dplyr/blob/bec50d3b1740db1fb28724e7e560cdf19924b97c/R/manip-dt.r)).

dplyr 0.5.0 (released June 2016) did not yet incorporate quasiqotation, so we can not run the above code in dplyr 0.5.0 unless we take the steps of:

* Replacing the !!x substitution markings with bquote()‘s .(x) notation.
* Changing the := back to =.
* Placing the whole pipeline in an eval(bquote()) block.
* Not directly substituting symbols on the left-hand sides of expressions.

This looks like the following.

# apply a partially parameterized pipeline using bquote

eval(bquote(

mtcars %>%

group\_by(.(group\_nm)) %>%

mutate(TMP1 = .(num\_nm)/.(den\_nm)) %>%

rename\_(.dots = setNames("TMP1", as.character(derived\_nm))) %>%

summarize(

TMP2 = mean(.(derived\_nm)),

TMP3 = n()

) %>%

ungroup() %>%

rename\_(.dots = setNames("TMP2", as.character(mean\_nm))) %>%

rename\_(.dots = setNames("TMP3", as.character(count\_nm))) %>%

arrange(.(group\_nm))

))

## # A tibble: 2 x 3

## am mean\_hp\_per\_cyl group\_count

##

## 1 0 22.7 19

## 2 1 23.4 13

That nearly solved the problem (notice we were not able to control the left-hand sides of assignment-like expressions). In December of 2016 we publicly announced and released the [let()](https://winvector.github.io/wrapr/reference/let.html) adapter that addressed almost all of these concerns at the user level (deliberately not modifying dplyr code). The substitutions are specified by name (instead of by position), and the code must still be executed in a let() block to get the desired control.

This looks like the following.

alias <-

c(group\_nm = as.character(group\_nm),

num\_nm = as.character(num\_nm),

den\_nm = as.character(den\_nm),

derived\_nm = as.character(derived\_nm),

mean\_nm = as.character(mean\_nm),

count\_nm = as.character(count\_nm))

wrapr::let(alias,

mtcars %>%

group\_by(group\_nm) %>%

mutate(derived\_nm = num\_nm/den\_nm) %>%

summarize(

mean\_nm = mean(derived\_nm),

count\_nm = n()

) %>%

ungroup() %>%

arrange(group\_nm)

)

## # A tibble: 2 x 3

## am mean\_hp\_per\_cyl group\_count

##

## 1 0 22.7 19

## 2 1 23.4 13

We felt this was the least intrusive way to give dplyr users useful macro substitution ability over dplyr pipelines. So by December of 2016 users had public access to a complete solution. More on wrapr::let() can be found [here](https://github.com/WinVector/wrapr/blob/master/extras/wrapr_let.pdf).

In May of 2017 rlang was publicly announced and released. dplyr adapted new rlang based quasiquotation ability directly into many of its functions/methods. This means the user does not need to draw a substitution block around their code, as the package authors have written the block. At first glance this is something only the dplyr package authors could do (as they control the code), but it turns out one can also easily adapt (or wrap) existing code in R.

A very small effort (about 25 lines of code now found in [bquote\_call()](https://github.com/WinVector/wrapr/blob/master/R/bquotefn.R)) is needed to specify argument substitution semantics using bquote(). The bulk of that code is adding the top-level := to = substitution. Then it takes only about 10 lines of code to write a function that re-maps existing dplyr functions to new functions using the bquote() adapter [bquote\_function()](https://github.com/WinVector/wrapr/blob/master/R/bquotefn.R). And a package author could choose to directly integrate bquote() transforms at even smaller cost.

Lets convert dplyr 0.5.0 to bquote() based quasiquotation. We can do that by running our adapter over the names of 29 common dplyr methods, which places bquote() enhanced versions in our running environment. The code to do this is below.

# wrap some dplyr methods

nms <- c("all\_equal", "anti\_join", "arrange",

"as.tbl", "bind\_cols", "bind\_rows",

"collect", "compute", "copy\_to", "count",

"distinct", "filter", "full\_join",

"group\_by", "group\_indices",

"inner\_join", "is.tbl", "left\_join",

"mutate", "rename", "right\_join",

"select", "semi\_join", "summarise",

"summarize", "tally", "tbl", "transmute",

"ungroup")

env <- environment()

for(fname in nms) {

fn <- getFromNamespace(fname, ns = "dplyr")

assign(fname,

# requires wrapr 1.6.4 or newer

wrapr::bquote\_function(fn),

envir = env)

}

With these adaptions in place we can write partially substituted or parametric code as follows.

packageVersion("dplyr")

## [1] '0.5.0'

# apply a parameterized pipeline using bquote

mtcars %>%

group\_by(.(group\_nm)) %>%

mutate(.(derived\_nm) := .(num\_nm)/.(den\_nm)) %>%

summarize(

.(mean\_nm) := mean(.(derived\_nm)),

.(count\_nm) := n()

) %>%

ungroup() %>%

arrange(.(group\_nm))

## # A tibble: 2 x 3

## am mean\_hp\_per\_cyl group\_count

##

## 1 0 22.7 19

## 2 1 23.4 13

This pretty much demonstrates that bquote() was up to the job the whole time. All one had to do is decide to incorporate it into the source code (something only the package author could do) or adapt the functions (a bit invasive, but something any user can do).

The user visible difference is that bquote() uses .(x) to mark substitution of x. rlang currently uses !!x to mark substitution of x (syntactically reminiscent of Lisp’s back-tick). Earlier versions of rlang used UQ(x), uq(x), and even ((x)) (signs of this can be found [here](https://github.com/r-lib/rlang/commit/61999344f025a40d24f49fdf31d403e63507edcf)). Other famous substitution notations include %x (from C/printf), {} (from [Python PEP 498 — Literal String Interpolation](https://www.python.org/dev/peps/pep-0498/), and used in [glue](https://cran.r-project.org/package=glue).), $ and ${} (from bash), and many [more](https://en.wikipedia.org/wiki/String_interpolation).

There are some minor technical differences between the bquote() and rlang solutions. But one can write corner case code that fails either method at will. We are not discussing the rlang !!! "splice" syntax, as the *need* for it is avoidable by avoiding over-use of "..." tricks. For actual user work both systems are going to look very similar.

Overall we are discussing avoiding interfaces that are convenient for interactive work, yet difficult to parameterize or program over. Quasiquotation is a tool to overcome package design limitations or design flaws. You don’t see quasiquotation very much in R, as many R developers work hard on their end to make sure the user does not require it.

For R package and function developers my advice is:

* If you are using "quoting interfaces" (interfaces that take names by quoting un-evaluated R-code) then design your package to also have sufficiently powerful standard evaluation interfaces so that users can move to these if they run into trouble. This [was the advice in the first edition of "Advanced R"](http://adv-r.had.co.nz/Computing-on-the-language.html):

As a developer, you should always provide an escape hatch: an alternative version of the function that uses standard evaluation.

* If you wish to introduce quasiquotation into your functions or packages, *please* consider using bquote() to do it. It has been part of base-R for 15 years, works very well, and is quite stable. bquote() is small enough to be comprehensible. Users who learn bquote() notation on one package will be able to re-use their experience on other bquote() enabled packages. To help: we have some example code called [bquote\_call\_args()](https://github.com/WinVector/wrapr/blob/master/R/bquotefn.R) where we are prototyping a canonical way to incorporate bquote() and := re-writing in your own functions and packages. It isn’t in the released version of wrapr yet, but it is testing well in the development version.