When explaining namespaces in a prior section, I noted the contents of one namespace are segregated from those of others. While this is certainly true, these objects must still exist somewhere, and so there should be a means of finding them. I actually only figured this out after an error message showed me the full ID for a namespaced object, but it has proven invaluable because there was one problem I had that could really only be solved by this method, or maybe making a mess of nested modules.

Remember back to the \*\*NS\*\* function and its two arguments. The first is \*\*namespace\*\* for the name of the namespace, and the second is \*\*id\*\* for some object. Normally you never need this second argument, as you create a function that takes that same \*\*id\*\* and creates the namespaced ID for you. This is what we do in the UI modules:

```{r echo = TRUE, eval = FALSE}

exampleUI <- function(name, VAL = 3) {

ns <- NS(name)

…

```

Rather than having to create another server module, like \*\*exampleCount\*\* to extract an object from a namespace, we are going to use \*\*NS\*\* to pierce inside a namespace and grab it directly.

```{ r echo = TRUE, eval = FALSE}

observe({

lapply(modList(), exampleServer)

output$modules <- renderUI( lapply(modList(), exampleUI) )

output$randCount <- renderText({

COUNT <- unlist(lapply(modList(), function(IN) input[[ NS(IN, "randCount") ]]))

paste0("There are ", sum(COUNT), " random numbers")

})

})

```

With \*\*NS\*\* we can get the complete namespaced ID of an object by giving it both the namespace name and the ID, in this case \*\*randCount\*\*. With this complete ID, we can select it from \*\*input\*\* using double brackets, and by doing this all within the function for \*\*lapply\*\*, the result will be a list of the \*\*randCount\*\* values for each namespace passed to that function.

Is this better than the method shown in the prior section? That probably comes down to your personal preference. Here the whole thing is encapsulated in a single line of code, or more specifically a single function within the call to \*\*lapply\*\*. Remember we still had to use \*\*lapply\*\* with the other method, but also create the \*\*exampleCount\*\* server module that is a larger block of code than this one.

The situation I originally applied this piercing to, involved a module that included a graph you could zoom into and a button. The purpose of the button was to take the zoom from the graph in one module, and apply it to the similar graphs in other instances of the same module. I could not find a way to achieve this same behavior with server modules, and while I could envision one, I saw it as a potential mess of nesting and interactions that I really did not want to deal with. But this piercing solution, having an \*\*observeEvent\*\* monitoring the pierced buttons, to then call server modules to apply the zoom to the other modular graphs was simple and pretty elegant too. Hence why I wanted to share it, as it is a useful tool to have.