```
Button("Run") {
  do { // calls to R can fail so there are lots of "try"s; poking at
less ugly alternatives

    // handling dots in named calls is a WIP
    _  = try R.evalParse("options(tidyverse.quiet = TRUE )")

    // in practice this wld be called once in a model
    try R.library("ggplot2")
    try R.library("hrbrthemes")
    try R.library("magick")

    // can mix initialiation of an R list with Swift and R objects
    let mvals: RObject = [
      "month": [ "Jan", "Feb", "Mar", "Apr", "May", "Jun" ],
      "value": try R.sample(100, 6)
    ]

    // ggplot2! `mvals` is above, `col.hexValue` comes from the color
picker
    // can't do R.as.data.frame b/c "dots" so this is a deliberately
exposed alternate call
    let gg = try R.ggplot(R.as_data_frame(mvals)) +
      R.geom_col(R.aes_string("month", "value"), fill: col.hexValue) +
// supports both [un]named
      R.scale_y_comma() +
      R.labs(
        x: rNULL, y: "# things",
        title: "Monthly Bars"
      ) +
      R.theme_ipsum_gs(grid: "Y")

    // an alternative to {magick} could be getting raw SVG from
{svglite} device
    // we get Image view width/height and pass that to {magick}
    // either beats disk/ssd round-trip
    let fig = try R.image_graph(
      width: Double(imageRect.width),
      height: Double(imageRect.height),
      res: 144
    )

    try R.print(gg)
    _ = R.dev_off() // can't do R.dev.off b/c "dots" so this is a
deliberately exposed alternate call

    let res = try R.image_write(fig, path: rNULL, format: "png")

    imgData = Data(res) // "imgData" is a reactive SwiftUI bound
object; when it changes Image does too
```

```
  } catch {
  }

}
```

that works in Swift as part of a SwiftUI app that displays a ggplot2 plot inside of a macOS application.

It doesn't shell out to R, but uses Swift 5's native abilities to interface with R's C interface.

I'm not ready to reveal that SwiftR code/library just yet (break's over and the core bits still need some tweaking) but I *can* provide some interim resources with an online book about working with R's C interface from Swift on macOS. It is uninspiringly called SwiftR — Using R from Swift.

There are, at present, six chapters that introduce the Swift+R concepts via command line apps. These aren't terribly useful (shebanged R scripts work *just* fine, #tyvm) in and of themselves, but command line machinations are a much lower barrier to entry than starting right in with SwiftUI (that starts in chapter seven).