Reading Property Lists in R With Swift

macOS relies *heavily* on property lists for many, many things. These can be plain text (XML) or binary files and there are command-line tools and Python libraries (usable via {reticulate}) that can read them along with the good 'ol `XML::readKeyValueDB()`. We're going to create a Swift function to read property lists and return JSON which we can use back in R via {jsonlite}.

This time around there's no need to create extra files, just install {swiftr} and your favorite R IDE and enter the following (expository is after the code):

```
library(swiftr)

swift_function(
  code = '

func ignored() {
  print("""
this will be ignored by swift_function() but you could use private
functions as helpers for the main public Swift function which will be
made available to R.
""")
}

@_cdecl ("read_plist")
public func read_plist(path: SEXP) -> SEXP {

  var out: SEXP = R_NilValue

  do {
    // read in the raw plist
    let plistRaw = try Data(contentsOf: URL(fileURLWithPath:
String(cString: R_CHAR(STRING_ELT(path, 0)))))

    // convert it to a PropertyList
    let plist = try PropertyListSerialization.propertyList(from:
plistRaw, options: [], format: nil) as! [String:Any]

    // serialize it to JSON
    let jsonData = try JSONSerialization.data(withJSONObject: plist ,
options: .prettyPrinted)

    // setup the JSON string return
    String(data: jsonData, encoding: .utf8)?.withCString {
      cstr in out = Rf_mkString(cstr)
    }

  } catch {
    debugPrint("\\(error)")
  }
  return(out)

}
')
```

This new `swift_function()` function — for the moment (the API is absolutely going to change) — is defined as:

```
swift_function(
  code,
  env = globalenv(),
  imports = c("Foundation"),
  cache_dir = tempdir()
)
```

where:

- `code` is a length 1 character vector of Swift code
- `env` is the environment to expose the function in (defaults to the global environment)
- `imports` is a character vector of any extra Swift frameworks that need to be `import`ed
- `cache_dir` is where all the temporary files will be created and compiled `dynlib` will be stored. It defaults to a temporary directory so specify your own directory (that exists) if you want to keep the files around after you close the R session

Folks familiar with `cppFunction()` will notice some (on-purpose) similarities.

The function expects you to expose only *one* `public` Swift function which also (for the moment) needs to have the `@_cdecl` decorator before it. You can have as many other valid Swift helper functions as you like, but are restricted to one function that will be turned into an R function automagically.

In this example, `swift_function()` will see `public func read_plist(path: SEXP) -> SEXP {` and be able to identify

- the function name (`read_plist`)
- the number of parameters (they all need to be `SEXP`, for now)
- the names of the parameters

A complete source file with all the `import`s will be created and a pre-built bridging header (which comes along for the ride with {swiftr}) will be included in the compilation step and a `dylib` will be built and loaded into the R session. Finally, an R function that wraps a `.Call()` will be created and will have the function name of the Swift function as well as all the parameter names (if any).

In the case of our example, above, the built R function is:

```
function(path) {
  .Call("read_plist", path)
}
```

There's a good chance you're using RStudio, so we can test this with it's property list, or you can substitute any other application's property list (or any `.plist` you have) to test this out:

```
read_plist("/Applications/RStudio.app/Contents/Info.plist") %>%
  jsonlite::fromJSON() %>%
  str(1)
## List of 32
##  $ NSPrincipalClass                     : chr "NSApplication"
##  $ NSCameraUsageDescription             : chr "R wants to access the
camera."
##  $ CFBundleIdentifier                   : chr "org.rstudio.RStudio"
##  $ CFBundleShortVersionString           : chr "1.4.1093-1"
##  $ NSBluetoothPeripheralUsageDescription: chr "R wants to access
bluetooth."
##  $ NSRemindersUsageDescription          : chr "R wants to access the
reminders."
##  $ NSAppleEventsUsageDescription        : chr "R wants to run
AppleScript."
##  $ NSHighResolutionCapable              : logi TRUE
##  $ LSRequiresCarbon                     : logi TRUE
##  $ NSPhotoLibraryUsageDescription       : chr "R wants to access the
photo library."
##  $ CFBundleGetInfoString                : chr "RStudio 1.4.1093-1, ©
2009-2020 RStudio, PBC"
##  $ NSLocationWhenInUseUsageDescription  : chr "R wants to access
location information."
##  $ CFBundleInfoDictionaryVersion        : chr "6.0"
##  $ NSSupportsAutomaticGraphicsSwitching : logi TRUE
##  $ CSResourcesFileMapped                : logi TRUE
##  $ CFBundleVersion                      : chr "1.4.1093-1"
##  $ OSAScriptingDefinition               : chr "RStudio.sdef"
##  $ CFBundleLongVersionString            : chr "1.4.1093-1"
##  $ CFBundlePackageType                  : chr "APPL"
##  $ NSContactsUsageDescription           : chr "R wants to access
contacts."
##  $ NSCalendarsUsageDescription          : chr "R wants to access
calendars."
##  $ NSMicrophoneUsageDescription         : chr "R wants to access the
microphone."
##  $ CFBundleDocumentTypes                :'data.frame':  16 obs. of
8 variables:
##  $ NSPhotoLibraryAddUsageDescription    : chr "R wants write access
to the photo library."
##  $ NSAppleScriptEnabled                 : logi TRUE
##  $ CFBundleExecutable                   : chr "RStudio"
##  $ CFBundleSignature                    : chr "Rstd"
##  $ NSHumanReadableCopyright             : chr "RStudio 1.4.1093-1, ©
2009-2020 RStudio, PBC"
##  $ CFBundleName                         : chr "RStudio"
##  $ LSApplicationCategoryType            : chr "public.app-category.
developer-tools"
##  $ CFBundleIconFile                     : chr "RStudio.icns"
##  $ CFBundleDevelopmentRegion            : chr "English"
```

FIN

A `source_swift()` function is on the horizon as is adding a ton of checks/validations to `swift_function()`. I'll likely be adding some of the SEXP and R Swift utility functions I've demonstrated in the [unfinished] book to make it fairly painless to interface Swift and R code in this new and forthcoming function.

As usual, kick the tyres, submit feature requests and bugs in any forum that's comfortable and stay strong, wear a 😷, and socially distanced when out and about.