R Shiny Apps have become a popular way of creating web applications in R. There are many ways of running Shiny Apps including locally in RStudio, on Shinyapps.io or installing the server software on your own host. I have been increasingly using Shiny apps as a way to demonstrate and interact with R Packages, especially packages I write for teaching purposes. Adding a Shiny app to an R package is relatively easy. In my use cases, I first put the application files (`server.R`, `ui.R`, and `global.R`) in the `inst/shiny` directory of my R package. I can then write a package function to run the Shiny app from the installed package directory using a function like this:

```
#' My Shiny App
#' @export
my_shiny_app <- function() {
    shiny::runApp(appDir = system.file('shiny',
package='MY_PACKAGE_NAME'))
}
```

This works very well when the entire app is self-contained. However, this does not work if you want to pass parameters to the Shiny app. In my situation, I want to be able to pass different data frames that I can interact with, but still have the Shiny app work if not parameters are passed. The first step to get this to work is to convert the `server.R` and `ui.R` scripts to functions within the R package. The code is largely the same, but instead of calling the functions we are going to assign them to `shiny_server` and `shiny_ui`, respectively. I have also included some minimal roxygen2 documentation. In particular, the functions need to be in the package's export file.

```
#' The Shiny App Server.
#' @param input input set by Shiny.
#' @param output output set by Shiny.
#' @param session session set by Shiny.
#' @export
shiny_server <- function(input, output, session) {
    if(!exists('thedata', envir = parent.env(environment()), inherits =
FALSE)) {
        message('thedata not available, using default faithful...')
        data(faithful, envir = environment())
        thedata <- faithful
    }

    output$environment <- renderPrint(
        print(ls(envir = parent.env(environment())))
    )

    output$thedata <- renderTable({
        return(thedata)
    })
}

#' The Shiny App UI.
#' @export
shiny_ui <- function() {
    fluidPage(
```

```
        titlePanel('Shiny Parameter Test'),
        verbatimTextOutput('environment'),
        tableOutput('thedata')
    )
}
```

This Shiny App doesn't do a lot. It has one user variable, `thedata`, and the user interface includes the output of `ls` (i.e. what is in the executing environment) and the contents of `thedata` (presumed to be a data frame). The important feature here is the first five lines of the `shiny_server`. I first check to see if `thedata` exists using the `!exists('thedata', envir = parent.env(environment()), inherits = FALSE)` command. In short, if `thedata` is not present, I want to set it to a reasonable default value.

When encapsulating the Shiny app in R scripts, using the `runApp` function with the `appDir` parameter is sufficient. In order to pass variables to the Shiny app, we need to control the environment the app is started in. Below, is a rewrite of the `my_shiny_app` app. First, we create a new environment that will contain all of our parameters. Since specifying the parameter is optional, we use the `missing` function to check to see if it has a value, and if so assign it to the new environment. We then set the environment to our server and ui functions the newly created environment that now contains our parameters. The rest is similar to creating Shiny apps in a single `app.R` file; create the app with the `shinyApp` function and start it with the `runApp` function, but with the app instead of a directory.

```
my_shiny_app <- function(thedata, ...) {
    shiny_env <- new.env()
    if(!missing(thedata)) {
        print('Setting parameters')
        assign('thedata', thedata, shiny_env)
    }
    environment(shiny_ui) <- shiny_env
    environment(shiny_server) <- shiny_env
    app <- shiny::shinyApp(
        ui = shiny_ui,
        server = shiny_server
    )
    runApp(app, ...)
}
```

We can now start the Shiny app with the `my_shiny_app()` function call. In the first instance, no parameters are passed to the app so the `faithful` data frame will be printed. The second and third calls will use the `iris` and `mtcars` data frames, respectively.

```
my_shiny_app()
my_shiny_app(thedata = iris)
my_shiny_app(thedata = mtcars)
```

The one disadvantage of this approach is that it is more difficult to run the Shiny app outside the package and maintaining the app in two formats is inconvenient. To address this issue the `save_shiny_app` function will save the server and ui functions in the package to a `server.R` and `ui.R` script files in the specified directory. Additionally, it will create a `global.R` file that loads the `shiny` package and any other required packages as specified in the `pkgs` parameter.

```
#' Save the Shiny App to ui.R, server.R, and global.R file.
```

```
#'
#' This function will create three files in the \code{out_dir}:
\code{server.R},
#' \code{ui.R}, and \code{global.R}. The contents of \code{server.R}
and
#' \code{ui.R} will be the source code of the \code{server_function}
and
#' \code{ui_function}, respectively. The \code{global.R} file will only
contain
#' \code{library} calls for \code{shiny} and any other packages
specified in
#' the \code{pkgs} parameter.
#'
#' If \code{run_app = TRUE} the function will start the Shiny app once
the
#' files are written. This is recommended to ensure all the necessary
packages
#' are loaded for the Shiny app to run.
#'
#' @param ui_function the function for the UI.
#' @param server_function the function for the server.
#' @param pkgs any packages that need to be loaded for the app to work.
At
#'        minimum the package containing the shiny app should be
included.
#' @param out_dir the directory to save the shiny app files.
#' @param run_app whether to run the app once the files are saved.
save_shiny_app <- function(ui_function,
                           server_function,
                           pkgs,
                           out_dir = 'shiny',
                           run_app = interactive()) {
    server_txt <- capture.output(server_function)
    ui_txt <- capture.output(ui_function)
    # Remove the bytecode and environment info
    server_txt <- server_txt[1:(length(server_txt)-2)]
    ui_txt <- ui_txt[3:(length(ui_txt)-3)]
    # Fix the function assignment
    server_txt[1] <- 'shinyServer(function(input, output, session)'
    server_txt[length(server_txt)] <- '})'
    global_txt <- c("library('shiny')")
    if(!missing(pkgs)) {
        global_txt <- c(global_txt, paste0("library('", pkgs, "')"))
    }
    # Save the shiny app files
    cat(server_txt, sep = '\n', file = paste0(out_dir, '/server.R'))
    cat(ui_txt, sep = '\n', file = paste0(out_dir, '/ui.R'))
    cat(global_txt, sep = '\n', file = paste0(out_dir, '/global.R'))
    # Start the app
    if(run_app) {
        runApp(appDir = out_dir)
    }
```

```
}
```