This tutorial is deprecated. Learn more about Shiny at our new location, shiny.rstudio.com.

Run & Debug

Throughout the tutorial you've been calling `runApp` to run the example applications. This function starts the application and opens up your default web browser to view it. The call is blocking, meaning that it prevents traditional interaction with the console while the application is running.

To stop the application you simply interrupt R – you can do this by pressing the Escape key in all R front ends as well as by clicking the stop button if your R environment provides one.

Running in a Separate Process

If you don't want to block access to the console while running your Shiny application you can also run it in a separate process. You can do this by opening a terminal or console window and executing the following:

```
R -e "shiny::runApp('~/shinyapp')"
```

By default `runApp` starts the application on port 8100. If you are using this default then you can connect to the running application by navigating your browser to http://localhost:8100.

Note that below we discuss some techniques for debugging Shiny applications, including the ability to stop execution and inspect the current environment. In order to combine these techniques with running your applications in a separate terminal session you need to run R interactively (that is, first type "R" to start an R session then execute `runApp` from within the session).

Live Reloading

When you make changes to your underlying user-interface definition or server script you don't need to stop and restart your application to see the changes. Simply save your changes and then reload the browser to see the updated application in action.

One qualification to this: when a browser reload occurs Shiny explicitly checks the timestamps of the ui.R and server.R files to see if they need to be re-sourced. If you have other scripts or data files that change Shiny isn't aware of those, so a full stop and restart of the application is necessary to see those changes reflected.

Debugging Techniques

Printing

There are several techniques available for debugging Shiny applications. The first is to add calls to the cat function which print diagnostics where appropriate. For example, these two calls to cat print diagnostics to standard output and standard error respectively:

```
cat("foo\n")
cat("bar\n", file=stderr())
```

Using browser

The second technique is to add explicit calls to the browser function to interrupt execution and inspect the environment where browser was called from. Note that using browser requires that you start the application from an interactive session (as opposed to using R -e as described above).

For example, to unconditionally stop execution at a certain point in the code:

```
# Always stop execution here
browser()
```

You can also use this technique to stop only on certain conditions. For example, to stop the MPG application only when the user selects "Transmission" as the variable:

```
# Stop execution when the user selects "am"
browser(expr = identical(input$variable, "am"))
```

Establishing a custom error handler

You can also set the R "error" option to automatically enter the browser when an error occurs:

```
# Immediately enter the browser when an error occurs
options(error = browser)
```

Alternatively, you can specify the recover function as your error handler, which will print a list of the call stack and allow you to browse at any point in the stack:

```
# Call the recover function when an error occurs
options(error = recover)
```

If you want to set the error option automatically for every R session, you can do this in your .Rprofile file as described in this article on R Startup.

← Previous Next →