How to build a Shiny app

ADDED: 06 JAN 2014

Let's walk through the steps of building a simple Shiny application. A Shiny application is simply a directory containing a user-interface definition, a server script, and any additional data, scripts, or other resources required to support the application.

UI & Server

To get started building the application, create a new empty directory wherever you'd like, then create empty `ui.R` and `server.R` files within in. For purposes of illustration we'll assume you've chosen to create the application at ~/shinyapp:

```
~/shinyapp
|-- ui.R
|-- server.R
```

Now we'll add the minimal code required in each source file. We'll first define the user interface by calling the function `pageWithSidebar` and passing it's result to the `shinyUI` function:

ui.R

```
library(shiny)

# Define UI for miles per gallon application
shinyUI(pageWithSidebar(

  # Application title
  headerPanel("Miles Per Gallon"),

  sidebarPanel(),

  mainPanel()
))
```

The three functions `headerPanel` , `sidebarPanel` , and `mainPanel` define the various regions of the user-interface. The application will be called "Miles Per Gallon" so we specify that as the title when we create the header panel. The other panels are empty for now.

Now let's define a skeletal server implementation. To do this we call `shinyServer` and pass it a function that accepts two parameters, `input and output` :

server.R

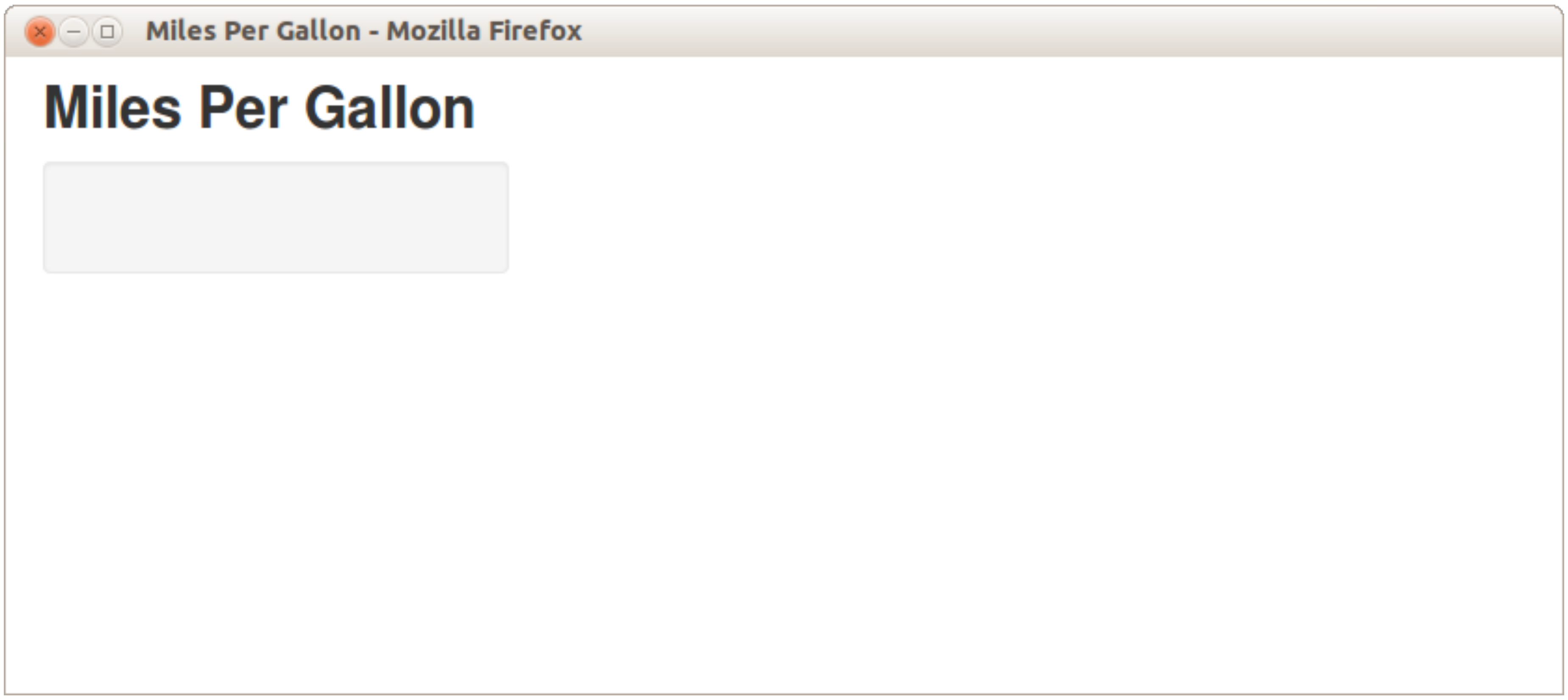
```
library(shiny)

# Define server logic required to plot various variables against mpg
shinyServer(function(input, output) {

})
```

Our server function is empty for now but later we'll use it to define the relationship between our inputs and outputs.

We've now created the most minimal possible Shiny application. You can run the application by calling the `runApp` function as follows:

```
> library(shiny)
> runApp("~/shinyapp")
```

If everything is working correctly you'll see the application appear in your browser looking something like this:



We now have a running Shiny application however it doesn't do much yet. In the next section we'll complete the application by specifying the user-interface and implementing the server script.

Inputs & Outputs

Adding Inputs to the Sidebar

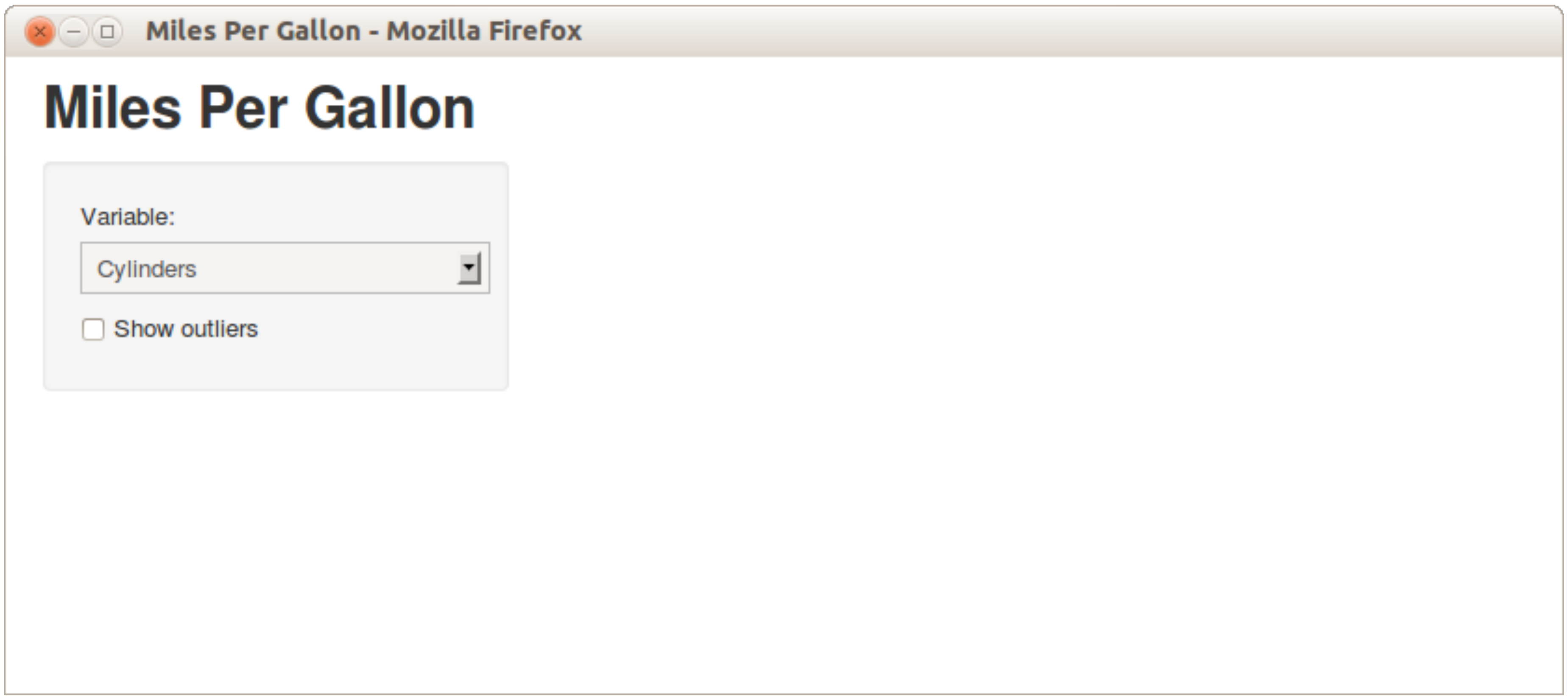
The application we'll be building uses the mtcars data from the R datasets package, and allows users to see a box-plot that explores the relationship between miles-per-gallon (MPG) and three other variables (Cylinders, Transmission, and Gears).

We want to provide a way to select which variable to plot MPG against as well as provide an option to include or exclude outliers from the plot. To do this we'll add two elements to the sidebar, a `selectInput` to specify the variable and a `checkboxInput` to control display of outliers. Our user-interface definition looks like this after adding these elements:

ui.R

```
library(shiny)

# Define UI for miles per gallon application
shinyUI(pageWithSidebar(

  # Application title
  headerPanel("Miles Per Gallon"),

  # Sidebar with controls to select the variable to plot against mpg
  # and to specify whether outliers should be included
  sidebarPanel(
    selectInput("variable", "Variable:",
                list("Cylinders" = "cyl",
                     "Transmission" = "am",
                     "Gears" = "gear")),

    checkboxInput("outliers", "Show outliers", FALSE)
  ),

  mainPanel()
))
```

If you run the application again after making these changes you'll see the two user-inputs we defined displayed within the sidebar:



Creating the Server Script

Next we need to define the server-side of the application which will accept inputs and compute outputs. Our server.R file is shown below, and illustrates some important concepts:

- Accessing input using slots on the `input` object and generating output by assigning to slots on the `output` object.
- Initializing data at startup that can be accessed throughout the lifetime of the application.
- Using a reactive expression to compute a value shared by more than one output.

The basic task of a Shiny server script is to define the relationship between inputs and outputs. Our script does this by accessing inputs to perform computations and by assigning reactive expressions to output slots.

Here is the source code for the full server script (the inline comments explain the implementation technqiues in more detail):

server.R

```
library(shiny)
library(datasets)

# We tweak the "am" field to have nicer factor labels. Since this doesn't
# rely on any user inputs we can do this once at startup and then use the
# value throughout the lifetime of the application
mpgData <- mtcars
mpgData$am <- factor(mpgData$am, labels = c("Automatic", "Manual"))

# Define server logic required to plot various variables against mpg
shinyServer(function(input, output) {

  # Compute the forumla text in a reactive expression since it is
  # shared by the output$caption and output$mpgPlot expressions
  formulaText <- reactive({
    paste("mpg ~", input$variable)
  })

  # Return the formula text for printing as a caption
  output$caption <- renderText({
    formulaText()
  })

  # Generate a plot of the requested variable against mpg and only
  # include outliers if requested
  output$mpgPlot <- renderPlot({
    boxplot(as.formula(formulaText()),
            data = mpgData,
            outline = input$outliers)
  })
})
```

The use of `renderText` and `renderPlot` to generate output (rather than just assigning values directly) is what makes the application reactive. These reactive wrappers return special expressions that are only re-executed when their dependencies change. This behavior is what enables Shiny to automatically update output whenever input changes.

Displaying Outputs

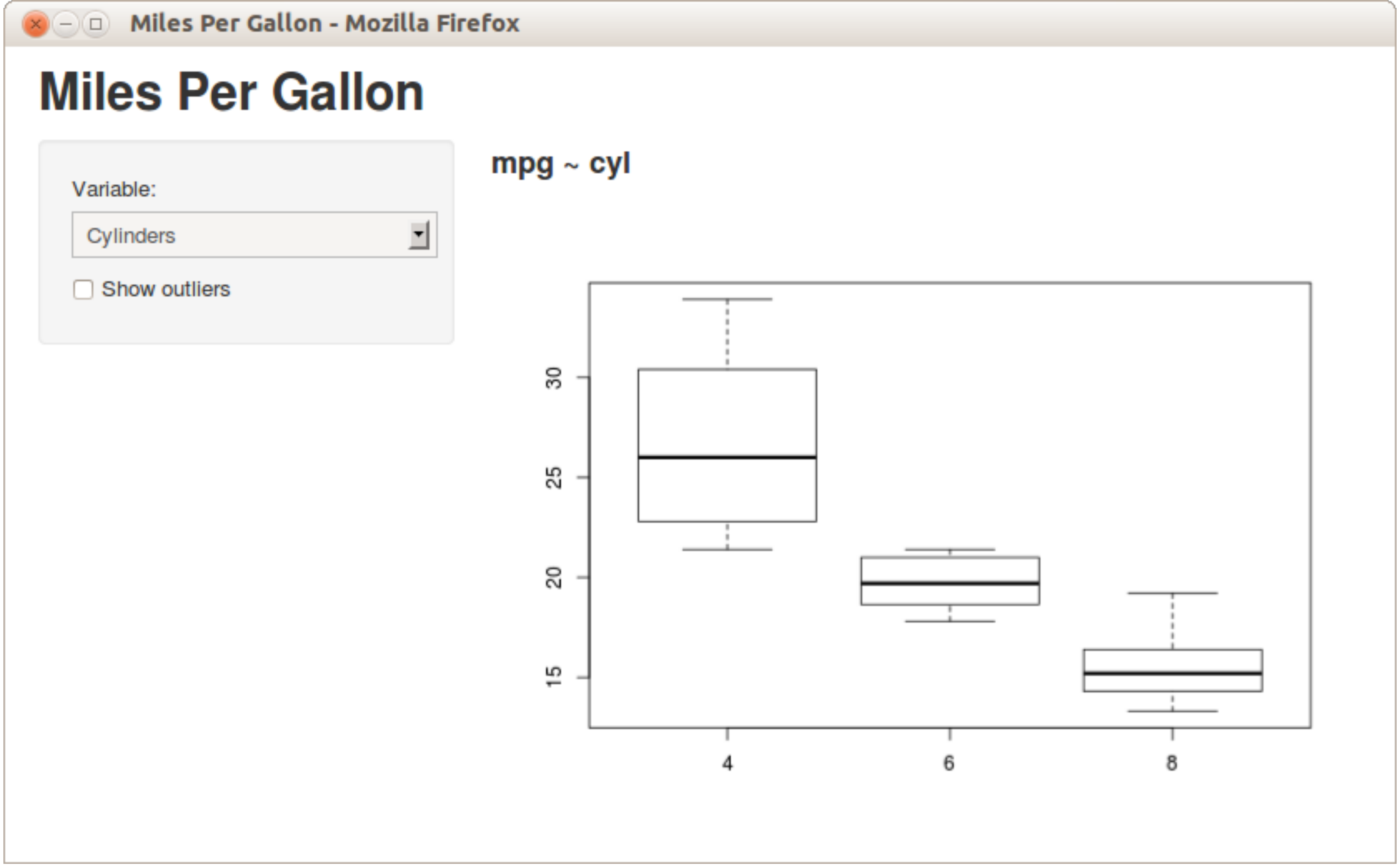
The server script assigned two output values: `output$caption` and `output$mpgPlot` . To update our user interface to display the output we need to add some elements to the main UI panel.

In the updated user-interface definition below you can see that we've added the caption as an h3 element and filled in its value using the `textOutput` function, and also rendered the plot by calling the `plotOutput` function:

ui.R

```
library(shiny)

# Define UI for miles per gallon application
shinyUI(pageWithSidebar(

  # Application title
  headerPanel("Miles Per Gallon"),

  # Sidebar with controls to select the variable to plot against mpg
  # and to specify whether outliers should be included
  sidebarPanel(
    selectInput("variable", "Variable:",
                list("Cylinders" = "cyl",
                     "Transmission" = "am",
                     "Gears" = "gear")),

    checkboxInput("outliers", "Show outliers", FALSE)
  ),

  # Show the caption and plot of the requested variable against mpg
  mainPanel(
    h3(textOutput("caption")),

    plotOutput("mpgPlot")
  )
))
```

Running the application now shows it in its final form including inputs and dynamically updating outputs:



Now that we've got a simple application running we'll probably want to make some changes. The next article (/articles/running.html) covers the basic cycle of editing, running, and debugging Shiny applications.

We love it when R users help each other, but RStudio does not monitor or answer the comments in this thread. If you'd like to get specific help, we recommend the Shiny Discussion Forum (https://groups.google.com/forum/#!forum/shiny-discuss) for in depth discussion of Shiny related questions and How to get help (http://shiny.rstudio.com/articles/help.html) for a list of the best ways to get help with R code.