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

GETTING STARTED
Welcome
Hello Shiny
Shiny Text
Reactivity

BUILDING AN APP
UI & Server
Inputs & Outputs
Run & Debug

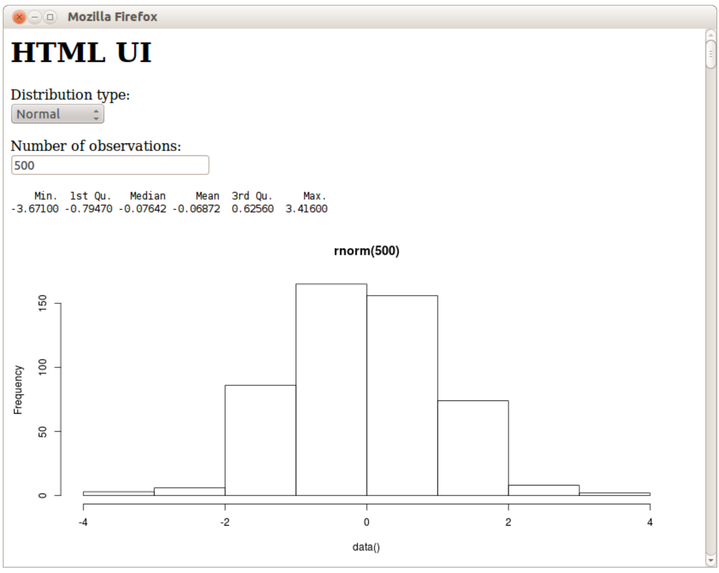
TOOLING UP
Sliders
Tabsets
DataTables
More Widgets
Uploading Files
Downloading Data
HTML UI
Dynamic UI

ADVANCED SHINY
Scoping
Client Data
Sending Images

UNDERSTANDING REACTIVITY
Reactivity Overview
Execution Scheduling
Isolation

DEPLOYING AND SHARING APPS
Deploying Over the Web
Sharing Apps to Run Locally

EXTENDING SHINY
Building Inputs
Building Outputs



The HTML UI application demonstrates defining a Shiny user-interface using a standard HTML page rather than a ui.R script. To run the example type:

```
> library(shiny)
> runExample("08_html")
```

Defining an HTML UI

The previous examples in this tutorial used a ui.R file to build their user-interfaces. While this is a fast and convenient way to build user-interfaces, some appliations will inevitably require more flexiblity. For this type of application, you can define your user-interface directly in HTML. In this case there is no ui.R file and the directory structure looks like this:

```
<application-dir>
|-- www
    |-- index.html
|-- server.R
```

In this example we re-write the front-end of the Tabsets application using HTML directly. Here is the source code for the new user-interface definition:

www/index.html

```
<html>

<head>
  <script src="shared/jquery.js" type="text/javascript"></script>
  <script src="shared/shiny.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="shared/shiny.css"/>
</head>

<body>
  <h1>HTML UI</h1>

  <p>
    <label>Distribution type:</label><br />
    <select name="dist">
      <option value="norm">Normal</option>
      <option value="unif">Uniform</option>
      <option value="lnorm">Log-normal</option>
      <option value="exp">Exponential</option>
    </select>
  </p>

  <p>
    <label>Number of observations:</label><br />
    <input type="number" name="n" value="500" min="1" max="1000" />
  </p>

  <pre id="summary" class="shiny-text-output"></pre>

  <div id="plot" class="shiny-plot-output"
       style="width: 100%; height: 400px"></div>

  <div id="table" class="shiny-html-output"></div>
</body>

</html>
```

There are few things to point out regarding how Shiny binds HTML elements back to inputs and outputs:

- HTML form elmements (in this case a select list and a number input) are bound to input slots using their `name` attribute.
- Output is rendered into HTML elements based on matching their `id` attribute to an output slot and by specifying the requisite css class for the element (in this case either shiny-text-output, shiny-plot-output, or shiny-html-output).

With this technique you can create highly customized user-interfaces using whatever HTML, CSS, and JavaScript you like.

Server Script

All of the changes from the original Tabsets application were to the user-interface, the server script remains the same:

server.R

```
library(shiny)

# Define server logic for random distribution application
shinyServer(function(input, output) {

  # Reactive expression to generate the requested distribution. This is
  # called whenever the inputs change. The output renderers defined
  # below then all used the value computed from this expression
  data <- reactive({
    dist <- switch(input$dist,
                   norm = rnorm,
                   unif = runif,
                   lnorm = rlnorm,
                   exp = rexp,
                   rnorm)

    dist(input$n)
  })

  # Generate a plot of the data. Also uses the inputs to build the
  # plot label. Note that the dependencies on both the inputs and
  # the data reactive expression are both tracked, and all expressions
  # are called in the sequence implied by the dependency graph
  output$plot <- renderPlot({
    dist <- input$dist
    n <- input$n

    hist(data(),
         main=paste('r', dist, '(', n, ')', sep=''))
  })

  # Generate a summary of the data
  output$summary <- renderPrint({
    summary(data())
  })

  # Generate an HTML table view of the data
  output$table <- renderTable({
    data.frame(x=data())
  })
})
```