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

Sending Images

When you want to have R generate a plot and send it to the client browser, the `renderPlot()` function will in most cases do the job. But when you need finer control over the process, you might need to use the `renderImage()` function instead.

About renderPlot()

`renderPlot()` is useful for any time where R generates an image using its normal graphical device system. In other words, any plot-generating code that would normally go between `png()` and `dev.off()` can be used in `renderPlot()`. If the following code works from the console, then it should work in `renderPlot()`:

```
png()
# Your plotting code here
dev.off()
```

```
# This would go in shinyServer()
output$myPlot <- renderPlot({
  # Your plotting code here
})
```

`renderPlot()` takes care of a number of details automatically: it will resize the image to fit the output window, and it will even increase the resolution of the output image when displaying on high-resolution ("Retina") screens.

The limitation to `renderPlot()` is that it won't send just any image file to the browser – the image must be generated by code that uses R's graphical output device system. Other methods of creating images can't be sent by `renderPlot()`. For example, the following won't work:

- Image files generated by the `writePNG()` function from the png package.
- Image files generated by the `rgl.snapshot()` function, which creates images from 3D plots made with the rgl package.
- Images generated by an external program.
- Pre-rendered images.

The solution in these cases is the `renderImage()` function.

Using renderImage()

Image files can be sent using `renderImage()`. The expression that you pass to `renderImage()` must return a list containing an element named `src`, which is the path to the file. Here is a very basic example of a Shiny app with an output that generates a plot and sends it with `renderImage()`:

server.R

```
shinyServer(function(input, output, session) {
  output$myImage <- renderImage({
    # A temp file to save the output.
    # This file will be removed later by renderImage
    outfile <- tempfile(fileext='.png')

    # Generate the PNG
    png(outfile, width=400, height=300)
    hist(rnorm(input$obs), main="Generated in renderImage()")
    dev.off()

    # Return a list containing the filename
    list(src = outfile,
         contentType = 'image/png',
         width = 400,
         height = 300,
         alt = "This is alternate text")
  }, deleteFile = TRUE)
})
```

ui.r

```
shinyUI(pageWithSidebar(
  headerPanel("renderImage example"),
  sidebarPanel(
    sliderInput("obs", "Number of observations:",
                min = 0, max = 1000,  value = 500)
  ),
  mainPanel(
    # Use imageOutput to place the image on the page
    imageOutput("myImage")
  )
))
```

Each time this output object is re-executed, it creates a new PNG file, saves a plot to it, then returns a list containing the filename along with some other values.

Because the `deleteFile` argument is `TRUE`, Shiny will delete the file (specified by the `src` element) after it sends the data. This is appropriate for a case like this, where the image is created on-the-fly, but it wouldn't be appropriate when, for example, your app sends pre-rendered images.

In this particular case, the image file is created with the `png()` function. But it just as well could have been created with `writePNG()` from the png package, or by any other method. If you have the filename of the image, you can send it with `renderImage()`.

Structure of the returned list

The list returned in the example above contains the following:

- `src`: The output file path.
- `contentType`: The MIME type of the file. If this is missing, Shiny will try to autodetect the MIME type, from the file extension.
- `width` and `height`: The desired output size, in pixels.
- `alt`: Alternate text for the image.

Except for `src` and `contentType`, all values are passed through directly to the `<img>` DOM element on the web page. The effect is similar to having an image tag with the following:

```
<img src="..." width="400" height="300" alt="This is alternate text">
```

Note that the `src="..."` is shorthand for a longer URL. For browsers that support the data URI scheme, the `src` and `contentType` from the returned list are put together to create a special URL that embeds the data, so the result would be similar to something like this:

```
<img src="
R4nOydd3ic1ZX/P2+ZKmlU"
  width="400" height="300" alt="This is alternate text">
```

For browsers that don't support the data URI scheme, Shiny sends a URL that points to the file.

Sending pre-rendered images with renderImage()

If your Shiny app has pre-rendered images saved in a subdirectory, you can send them using `renderImage()`. Suppose the images are in the subdirectory `images/`, and are named `image1.jpeg`, `image2.jpeg`, and so on. The following code would send the appropriate image, depending on the value of `input$n`:

server.R

```
shinyServer(function(input, output, session) {
  # Send a pre-rendered image, and don't delete the image after sending it
  output$preImage <- renderImage({
    # When input$n is 3, filename is ./images/image3.jpeg
    filename <- normalizePath(file.path('./images',
                              paste('image', input$n, '.jpeg', sep='')))

    # Return a list containing the filename and alt text
    list(src = filename,
         alt = paste("Image number", input$n))

  }, deleteFile = FALSE)
})
```

In this example, `deleteFile` is `FALSE` because the images aren't ephemeral; we don't want Shiny to delete an image after sending it.

Note that this might be less efficient than putting images in `www/images` and emitting HTML that points to the images, because in the latter case the image will be cached by the browser.

Using clientData values

In the first example above, the plot size was fixed at 400 by 300 pixels. For dynamic resizing, it's possible to use values from `session$clientData` to detect the output size.

In the example below, the output object is `output$myImage`, and the width and height on the client browser are sent via `session$clientData$output_myImage_width` and `session$clientData$output_myImage_height`. This example also uses `session$clientData$pixelratio` to multiply the resolution of the image, so that it appears sharp on high-resolution (Retina) displays:

server.R

```
shinyServer(function(input, output, session) {

  # A dynamically-sized plot
  output$myImage <- renderImage({
    # Read myImage's width and height. These are reactive values, so this
    # expression will re-run whenever they change.
    width  <- session$clientData$output_myImage_width
    height <- session$clientData$output_myImage_height

    # For high-res displays, this will be greater than 1
    pixelratio <- session$clientData$pixelratio

    # A temp file to save the output.
    outfile <- tempfile(fileext='.png')

    # Generate the image file
    png(outfile, width=width*pixelratio, height=height*pixelratio,
        res=72*pixelratio)
    hist(rnorm(input$obs))
    dev.off()

    # Return a list containing the filename
    list(src = outfile,
         width = width,
         height = height,
         alt = "This is alternate text")
  }, deleteFile = TRUE)

  # This code reimplements many of the features of `renderPlot()`.
  # The effect of this code is very similar to:
  # renderPlot({
  #   hist(rnorm(input$obs))
  # })
})
```

The `width` and `height` values passed to `png()` specify the pixel dimensions of the saved image. These can differ from the `width` and `height` values in the returned list: those values are the pixel dimensions to used display the image. For high-res displays (where `pixelratio` is 2), a "virtual" pixel in the browser might correspond to 2 x 2 physical pixels, and a double-resolution image will make use of each of the physical pixels.