Dynamic and Reactive Web-based Graphics for R

Simon J. Potter
simon.potter@auckland.ac.nz

February 18, 2013

A collection of tools has been developed to create and present web-based graphics via the popular statistical software, R. The primary motivation for this is that web browsers have become a powerful and flexible tool for building rich user interfaces. They feature many capabilities that are difficult, if not impossible to do in R. Many packages, notably RStudio's shiny, are able to present a reactive user interface based on input from users. However, no R packages are able to provide a way of generating reactive *graphics* from R. Instead, entirely new plots must be drawn to show changes in state, rather than being modified themselves.

The gridSVG package exports grid plots as SVG images. It has been extensively modified to support the goal of generating reactive graphics. In the past, it has been able to generate images that were capable of animation and interactivity (via JavaScript event handlers and hyperlinking). However, it did not easily allow manipulation of the image after it had been created. To correct this, two major features have been added; exporting of a gridSVG coordinate system and node based SVG document generation.

The node based approach to drawing SVG is a result of using the XML package. The two main reasons for rewriting to use XML are because of the ability to construct in-memory images, and to insert, remove or modify XML nodes at any location within our SVG document. Drawing images in memory allows us to more easily serve gridSVG images over the web as it avoids the need to touch a hard disk for storage, as is currently the case for all graphics devices not dependent on a GUI. The ability to insert nodes at any location is particularly useful if we wish to modify the image, as we can add, remove or modify an SVG element, and its children. The biggest advantage of using XML is we can *modify* an image after it has been drawn. An example of such behaviour would be to remove, add, or alter SVG content to customise the resulting plot.

The exporting of grid's coordinate system allows us to re-use a grid viewport even after it has been exported via gridSVG. In JavaScript, this is useful because we are able to add extra graphic objects to a plot, relative to a scale, without the need to use R. The exported coordinate information can also be imported into R to generate parts of a plot, allowing one to modify an existing plot.

Commonly in a web browser, particularly with any of the popular JavaScript libraries, we are able to programmatically select content in a web page using CSS3 Selectors. These are often very readable and terse expressions that make it easy for developers to target particular pieces in a web page. It would be ideal if we could specify parts of an XML document (using the XML package) with the same convenient notation. The XML package does provide a notation that performs this task, called XPath. However, it is not commonly

employed within web browsers and its usage is limited to those familiar with a web-based environment. A package has been created, selectr, which translates CSS3 Selectors into XPath expressions, allowing us to use the same notation within R (with XML) as we would in a web browser. Convenience functions have also been created that mimic the `querySelector()` method present in a web browser. This method retrieves matching nodes based on a CSS selector. The creation of the selectr package eases manipulation of HTML and XML documents in R.

If we have an image within the browser that we wish to animate, we first need to determine how the image is to be animated. A problem with animation is that actions are often influenced by previous animations, and thus need to be sequenced in a particular order. Using a JavaScript library like D3 (`http://d3js.org/`), we would have to hard-code the sequencing ourselves, but this has limitations. The first of these limitations is that we have to keep track of the relationships between animations ourselves. For example, consider a sequence of animations that happens one after the other. If one of the earlier animations requires a longer period of time, it will affect any following animations. Without any animation management, we would need to modify all following animations to account for this.

To manage these sequences of animations, the animaker package was created. It allows us to describe in R how an animation should be arranged. The result of this description is that it allows us to export the times and durations of each animation, along with the context in which the animation occurred. The context gives us information such as how many times the animation has been repeated within a sequence. If we export this information to JSON via the RJSONIO package, then this information is available to us in JavaScript. We can use this information to build a sequence of animations without being concerned about when they occur, or for how long.

The advantage of each of these packages is that they can be used independently. Although they can be combined to produce dynamic and reactive graphics, each package is useful in its own right. This is particularly important because we can produce these graphics regardless of the underlying mechanism used to send information to and from R. Therefore we could use packages such as websockets, shiny, Rook, etc. without being limited in the type of graphics we can produce. This means that by combining the browser with R, we are able to produce interactive graphics that can *react* to changes in the browser. As a result animation can be applied to these changes, clearly demonstrating the visual effects of a change in state. The development of these packages allows us to be able to produce *reactive* graphics that are both interactive and able to be animated.