Table of Contents

Motivation

The idea is that R programmers can create JavaScript functions for use in SVG, Flash and HTML documents for Google Maps, Google Earth, and generally. While they can write the code directly in JavaScript, and this is best, for some or at certain times, it may be convenient to write the JavaScript code as if it were R and to have the R code be "compiled" to JavaScript code. For example, if we want to end up a with a JavaScript function that looks like[1]

R
```
function setLambda(evType, group, val)
{
        /* If it is the same value, don't do anything*/
    if(Math.floor(val) == cur_lambda)
 return(0);
    setVisibility(cur_lambda, 'hidden', numPoints);
    cur_lambda = Math.floor(val);
    setVisibility(cur_lambda, 'visible', numPoints);
}
```

The corresponding R code could be written as

R
```
setLambda =
function(evType, group, val)
{
       # If it is the same value, don't do anything
    if(Math.floor(val) == cur_lambda)
 return(0)
    setVisibility(cur_lambda, 'hidden', numPoints)
    cur_lambda = Math.floor(val)
    setVisibility(cur_lambda, 'visible', numPoints)
}
```

Okay, so that one is entirely trivial, but it illustrates the point. We can compile this by adding ';' to the end of each statement.

Let's look at a different function, or in this case expression.

R
```
el = document.getElementById("residual-group-" + lambda);
```

Again, we can leave this as is. Since we don't actually evaluate this, the string concatenation, "residual-group-" + lambda, does not cause a problem.

[1]Taken from linkedSmoother.js in SVGAnnotation/tests

To increase semantic information so that we can do analysis in R of the function, we might want to write method invocation as

R
```
document$getElementById("residual-group" + lambda)
```

The $ operator indicates a call to the method getElementById in the object document.

Similarly, the [[operator can be used for accessing attributes. For example,

R
```
obj[["visible"]]
```

would correspond to obj.visible in JavaScript. We could use $ for both as we can recognize a method invocation when processing the R code.

What about loops? Consider the JavaScript code

R
```
for(i = 0 ; i < numPoints; i++) {
    el = document.getElementById("residual-" + lambda + "-" + (i+1));
    el.setAttribute('visibility', state);
}
```

We can recognize R code that loops over a sequence object and map it to the JavaScript code above. For example,

R
```
for(i in seq(along = obj)) {...}
for(i in seq(length = numPoints)) {...}
for(i in 1:numPoints) {...}
```

Keywords

The keywords are var, if, switch, for, while, do, break, continue, return, try, throw, and with

++, --, +=

instanceof

null, undefined, NULL

The switch mechanism in JavaScript can be mimiced with the `switch`() function in R.

The JavaScript return is richer than R's and can return an associative array in a different form

R
```
return {
    event: event,
    op: event.type,
    to: event.srcElement,
    x: event.clientX + document.body.scrollLeft,
    y: event.clientY + document.body.scrollTop};
```

(See http://www.crockford.com/javascript/survey.html)

JavaScript Constructors of the form new Class(a, b); can be written as `new("Class", a, b)`

We can create empty associative arrays with

R
```
foo = {}
```
We have to recognize this as being the empty expression block.

Empty arrays are trickier as R won't parse `[]`.

Variables in R

We can serialize R objects into the JavaScript using RJSONIO or rjson. Further, we can analyze the R code and rewrite it, but also substitute particular symbols with their R values. For example, suppose we want JavaScript code such as

R
```
var sliderStyles= {"stroke":"dimgray", "stroke-width":3};
  var invisSliderWidth = 15;
  slider1 = new slider("alpha", "slider-alpha", 100, 510, 1, 475, 510,
                        len_a, 1, sliderStyles, invisSliderWidth,
                         "sliderSymbol", showVal, true);
```
in which showVal and slider1 refer to a JavaScript function and variable defined elsewhere. Then we might write this in R as

R
```
function(evt, len_a, len_b) {
  sliderStyles= tb;
  invisSliderWidth = 15;
  slider1 = new slider("alpha", "slider-alpha", 100, 510, 1, 475, 510,
                        len_a, 1, sliderStyles, invisSliderWidth,
                         "sliderSymbol", showVal, true);
   ...
}
```
We have specified the initialization vale of sliderStyles with another variable. When we "compile" this, we can specify which R variables to substitute into the JavaScript code with, e.g.,

R
```
jsCompile(code,
          substitute = list(tb = c(stroke = "dimgray",
                                   'stroke-width' = 3)),
           jsglobals = c(slider1, showVal))
```

Cons

This may not be a good idea. People should program JavaScript in JavaScript. The difficult lies in that we can't run JavaScript embedded in R (for graphical content at least). We have data in R and we want to export that. We can do this with RJSONIO, but we have to write the JavaScript code separately or use a lot of string manipulation.