Tutorial

By: Elisha Emond

IT704

What is Raphael JS?

This is a small graphics JavaScript library which was designed to simplify work with drawing vector graphics in your browser. It uses the SVG W3C Recommendation and VML as its base for creating graphics so every object you create is a DOM object, which means JavaScript event handlers can be attached and they can be modified whenever. This library makes drawing vector art cross-browser easy, and on top of that it's pretty easy to use!

Materials necessary to start using Raphael:

- Raphael downloaded (v. 2.1.2) www.raphaeljs.com – What is recommended is to GZIP it to reduce file size however there is an option to download the uncompressed source as an alternative.
- A understanding of JavaScript and the power of DOM manipulation (though not necessary to know to understand how to use Raphael) – read about it on www.w3schools.com/js/js_intro.asp
- Text editor of your choice – Recommended: Notepad++

Getting started with Raphael:

When starting off with a project in Raphael you must first create an HTML doc in your text editor which includes a path to your script.js file which will be where all your JavaScript code is. Also it will in include the path to your Raphael download as well as a div containing an id equal to your container which will later be used when referencing your canvas (the canvas is where all Raphael objects are drawn onto). Your HTML file should look similar to this:

```
<!DOCTYPE HTML>
<html lang = “en”>
    <head>
        <title>Raphael Tut</title>
        <script type="text/javascript" src="path/to/raphael.js"></script>
        <script type="text/javascript" src="path/to/script.js"></script>
    </head>
    <body>
        <div id="container"></div>
    </body>
</html>
```

Once you have all of this set up your next step is to create a Raphael canvas object which will be done inside your script.js file. This “canvas” is created using a Raphael() object and there are a couple different ways to do it depending on how you have the rest of your code setup however I’m going to focus on the way in which the Raphael canvas is tied to the HTML using an HTML element, in the case of the above example it would be the id = container. The canvas will be drawn inside of this element and it should be drawn upon page load. An example of a canvas being drawn would be:

```
window.onload = function() {

var paper = new Raphael(document.getElementById('container'), 500, 500);

}
```

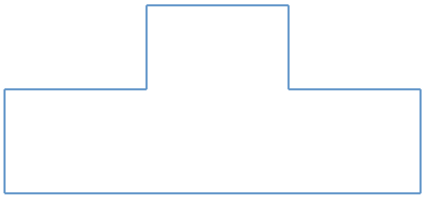
In this case because the Raphael canvas is called “paper” which in most if not all cases it is called by developers (you’ll notice even in the Raphael documentation it is referred to as this), all Raphael objects will be referred to by using it. For example if you wish to draw a circle at coordinates 30x30 with a radius of 8 you would do so like this:

```
var circle = paper.circle(30, 30, 8);
```

Many different shapes can be drawn using Raphael including circles as shown above, rectangles, ellipses, and lines can be drawn to make any other shape you desire. For example one that isn't a pre-coded Raphael shape is a tetronimo which can be drawn with a line like this:

```
var tetronimo = paper.path("M 250 250 l 0 -50 l -50 0 l 0 -50 l -50 0 l 0 50 l -50 0 l 0 50 z");
```

The shape would look something like this (not drawn to scale):



Shape Animation:

Shapes can be animated in a variety of ways. You first create an animation object which for example you could write this line of code:

```
var anim = Raphael.animation({cx: 15, cy: 25}, 2e3);
```

The animation you want to have is specified within the parenthesis of the code above when creating your animation object. In the case of this example it will cause the center of objects to move from their current location to the coordinate (15, 25) within your canvas. To animate your shapes just call Element.animate() on the desired shapes and inside the parenthesis place the name of your variable, in the case of the example above it would be anim. You can delay animations of objects by any number of milliseconds desired by adding .delay(#milliseconds) onto your animation variable. Also, you can repeat animations by adding .repeat(#times) onto your animation variable when animating an object. An example of animating an object named circle with a delay of 500ms would be:

```
circle.animate(anim.delay(500));
```

Another function that can be called on your shape objects is the animateWith() function which ensures that your shapes will run in sync with each other if that is what you wish them to do.

Shape manipulation:

You can do a multitude of things with the shapes you create with this library from calling events to getting various attributes of objects. Inside the Raphael documentation each of the available functions that can be used are preceded with Element. and are in the "Element" section. One thing you can do to an object is clone it, which makes a copy of whatever shape you specify. This is done by calling Element.clone() on the name of the shape you wish to copy. You can change various attributes of your objects using the Element.attr() function such as the font of a string variable (yes you can have text put into your canvas! Do this using Paper.text(x, y, "text") x and y being the coordinates of where you desire the text to be placed) or the color of a circle from red to blue by changing the fill attribute.

Elements can be dragged, have something happen when they're hovered over or being clicked/double clicked, and you can even hide them say for example if you wanted to hide them after showing them for a certain amount of time. Raphael also utilizes the mouse event functions such as mousemove() etc. You can send elements to the front or back of the rest in the canvas, rotate them, etc. Others can be found on the Raphael documentation page on their website.

g.Raphael:

So I've discussed some neat Raphael stuff, and animating shapes is fun but what about how this library can be used in practical ways? While neat animations can be done and fun charts can be made using plain old Raphael, there is actually an extension of Raphael that can be used

to create more formal charts. Raphael can be used to make the same kinds of charts however there are only interactive options for dot charts and line charts. g.Raphael is what the extension is called and can be downloaded in the same way as Rapahel from the site:

http://g.raphaeljs.com/

The developers of Raphael created this so that people could put interactive charts onto their websites. g.Raphael has to be included into your HTML document in the same way you include Raphael. The charts which g.Raphael supports are line, pie, dot, and bar charts/graphs. If you go to the link listed above there are several demos of these charts right on the landing page. To use any of these charts however you must download the js file for them and also include it into your HTML document so for example if you want to create a pie chart you would need to have g.pie.js downloaded and included along with g.raphael.js and raphael.js.

When using g.Raphael you still need to have a canvas drawn with Raphael inside your script.js file (the file which holds your JavaScript/Raphael code). And using that canvas variable, say "r" you can create a pie chart with the center at coordinates (320,200), a radius of 100, and split by the data 55, 20, 13, 32, 5, 1, and 2 the code would look like:

```
r.piechart(320, 240, 100, [55, 20, 13, 32, 5, 1, 2]);
```

To Conclude:

This tutorial/document should get you started with Raphael.js pretty quick, the Raphael site makes it pretty easy to understand and if you've ever coded in say Java and manipulated objects with things like the drag or click events it will be all too familiar when it comes to animating objects. The g.Raphael extension for Raphael is really a great tool for if you want an interactive chart on your website. It's pretty easy to use and setting it up is really just downloading the necessary files for your project and including them accordingly. Honestly if you

know what you want to create it's pretty easy to make it, and there are so many examples online so if you get stuck or need some fresh ideas it's as easy as looking through some examples people have posted and checking out the source code. After all it is a JavaScript library so you can see everything the developer of the object/chart made when you view the source code in your browser.

References:

http://raphaeljs.com

http://code.tutsplus.com/tutorials/an-introduction-to-the-raphael-js-library--net-7186

http://g.raphaeljs.com/