Contents

Introduction

Having used Qlikview for many years, I decided it was time to learn about extensions.
I come from a financial background with some SQL and no javascript or web experience, so I am pretty much starting from scratch.

As with many things Qlik, I found the information to be very sparse, hard to track down and sometimes incredibly technical which means it can take a very long time to get to a point where it all makes sense and I nearly gave up a couple of times.

So I've referenced my sources here to bring everything in to one place and this absolutely doesn't replace any of the sources I have used, it just collects and expands on them. Without these excellent articles I would never had learned how to bring all this together.

Selfishly this is also great for me as even as I have been writing this I've learned extra bits about what I have done and how things work.

The first two parts of this tutorial are published here on the community, will the remaining chapters on my blog www.qlikanddirty.com which you can of course access openly.

Steps to follow

Firstly download some D3 code, open it up in notepad++ and try and hack it into a Qlik extension then get annoyed it doesn't work.
Ok well that is how I started, then I took a deep breath and realised that this probably wasn't something I was going to be able to learn by hacking pieces of code together from google . So I decided to build my extension up piece by piece so I could digest everything and learn what each bit did and how it interacts with Qlik.

So the sensible steps to follow (depending on your existing knowledge) are:

1. Take the Javascript Course on code academyJavaScript | Codecademy
2. Fill in that knowledge with the css bits from w3schools JavaScript Tutorial
3. Work through the excellent examples by barryharmsen on QlikView Extension tutorials, documentation and examples - The Qlik Fix! The Qlik Fix!
4. Learn to love the extension template by bmz on Object Extension Templates
5. Make my tweaks to the extension template (See attached file)
6. Go Nuts with D3

When you've worked through steps 1-3 and digested step 4 then this is where this tutorial fits in.

The Extension Template Script.js (adjusted)

Before we try anything, lets get to grips with this extension template. If you look at the code off the bat its going to scare the %£$ out of you, but actually Brian explains it very well and I've expanded on this below with the little tweaks I have added or where I think a little explanation is needed.
Run the qar file and it will extract the template to your extensions folder and then navigate to it and open up the Script.js file.

The Name, Path and Files Required

The first two lines just set up some simple variables for the extension name and path

```
var extensionName = "_Adams Template File"; //remember to also update this
in definition.xml
var extensionPath = Qva.Remote + "?public=only&name=Extensions/" +
extensionName +"/";
```

I slightly changed the template so you only have to enter your extension name in one place, as you will see from my comments this is because I spent hours looking for a simple / which I had missed from the extension path.
It also means when we add the actual extension code, we don't have to type the name again.

The next step is to load the additional JavaScript files which you might want, again I changed this purely as I didn't like the nesting of functions, I think this just reads a bit easier:

```
function extensionInit() {
  var jsfiles = [];
  //load jquery if required
  if (typeof jQuery == 'undefined') {
  jsfiles.push(extensionPath + "jquery.js");
  }
  //Pushing any other js files
  jsfiles.push(extensionPath + "d3.v3.min.js");
  //jsfiles.push(extensionPath + "additionalfileshere.js");

  //now load these and call next function
  Qva.LoadScript(jsfiles, extensionDone);
  } //end extension_Init
```

So rather than nesting Qva.LoadScript functions, we just push the files we want into an array and then call this.
Breaking down the code by way of explanation:

Just declare the function and then set a variable ([ ] means blank array) to hold the script names

```
function extensionInit() {
  var jsfiles = [];
```

This is standard code to check if jQuery is loaded or not and if not then it loads this as the first items into our array.

```
if (typeof jQuery == 'undefined') {
  jsfiles.push(extensionPath + "jquery.js");
  }
```

Then we can just repeat the push line for any other files that we want to load. In this case we want to load the d3 script.

```
  jsfiles.push(extensionPath + "d3.v3.min.js");
```

Finally we just tell qlikview to load the files and then call the next function to run which is the extension function.

```
  Qva.LoadScript(jsfiles, extensionDone);
  } //end extension_Init
```

Then a Big Chunk of Confusing Code

Ignore it! I did after figuring out that it is basically required to enable click events in the extension. I just took it as wrote and left it well alone.

```
if (Qva.Mgr.mySelect == undefined) {
    Qva.Mgr.mySelect = function (owner, elem, name, prefix) {
        if (!Qva.MgrSplit(this, name, prefix)) return;
        owner.AddManager(this);
        this.Element = elem;
        this.ByValue = true;
        elem.binderid = owner.binderid;
        elem.Name = this.Name;
        elem.onchange = Qva.Mgr.mySelect.OnChange;
        elem.onclick = Qva.CancelBubble;
    }
    Qva.Mgr.mySelect.OnChange = function () {
        var binder = Qva.GetBinder(this.binderid);
        if (!binder.Enabled) return;
        if (this.selectedIndex < 0) return;
        var opt = this.options[this.selectedIndex];
        binder.Set(this.Name, 'text', opt.value, true);
    }
    Qva.Mgr.mySelect.prototype.Paint = function (mode, node) {
        this.Touched = true;
        var element = this.Element;
        var currentValue = node.getAttribute("value");
        if (currentValue == null) currentValue = "";
        var optlen = element.options.length;
        element.disabled = mode != 'e';
        //element.value = currentValue;
        for (var ix = 0; ix < optlen; ++ix) {
            if (element.options[ix].value === currentValue) {
                element.selectedIndex = ix;
            }
        }
        element.style.display = Qva.MgrGetDisplayFromMode(this, mode);
    }
}
```

Loading the Extension Code

This is where the main code gets loaded, there is only one small change I made here and that is to use the extension name variable we set as the first line, rather than having to retype it (KISS!)

```
Qva.AddExtension(extensionName , function(){
```

Finally Some Debugging

I REALLY like this little bit of code thank you dgudkov BI Review: Building extensions in QlikView: some hints & tips Firebug is one of the best debugging tools and its available as a firefox add-on, however with this little bit of code you can run it in any browser, just hit F12 when in the document on the access point.

However it does also present within the Qlikview application. While it can't read the script files to help debugging (Javascript can't load local files) it does display the console which is really useful as you can console.log() messages to test things.

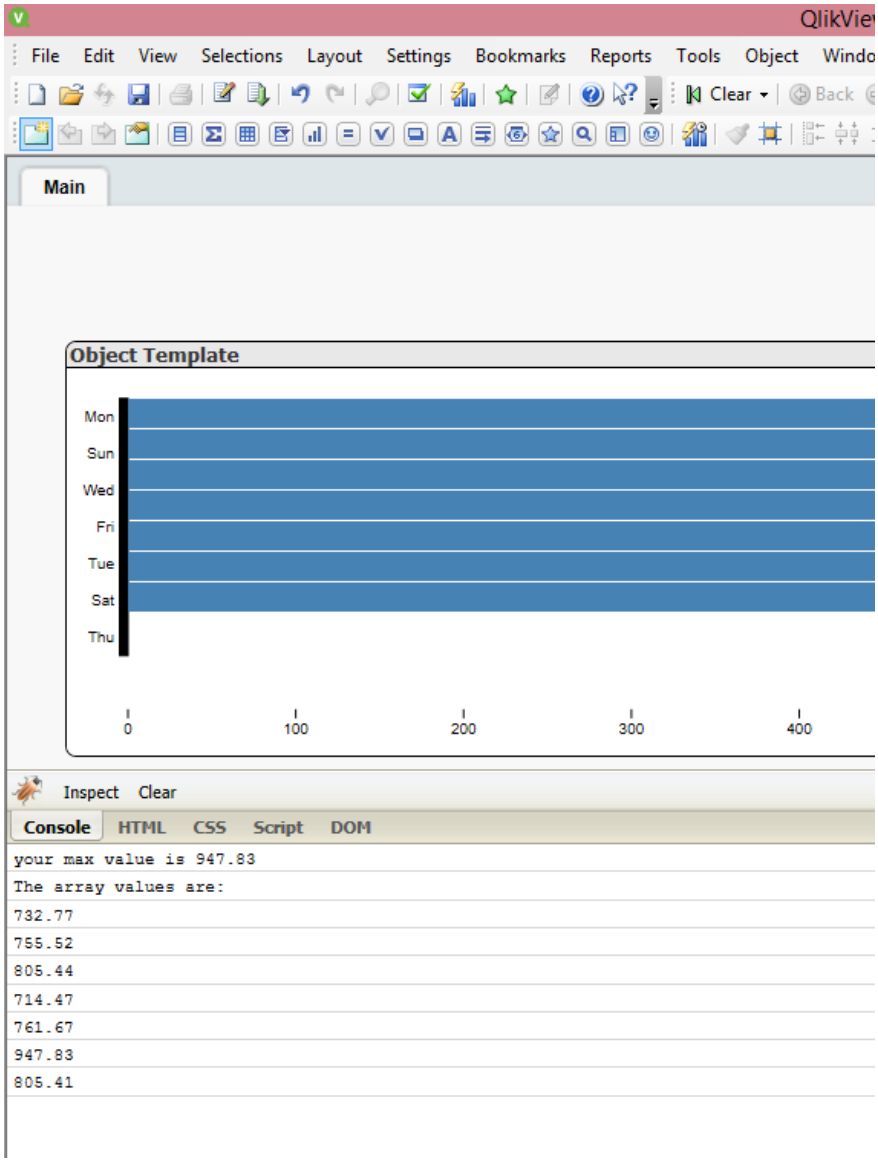
This is a slight tweak on the way Dmitry uses it, again for simplicity's sake I found it easier just to run the extensionInit function inside it as that is the function which everything runs from

```
Qva.LoadScript('https://getfirebug.com/firebug-lite.js', function(){
extensionInit();
});
```

For example I wanted to double check the values in an array so I used the following code (you can use window.alert etc as well):

```
console.log("your max value is "+d3.max(dataset));
console.log("The array values are:");
for (var b=0; b< dataset.length; b++){
console.log(dataset[b]);
}
```

And there you go, right there in your application you've got the console window showing you your debugging values:


QlikVie
File Edit View Selections Layout Settings Bookmarks Reports Tools Object Windo
Clear
Back
Main
Object Template
Mon
Sun
Wed
Fri
Tue
Sat
Thu
0
100
200
300
400
Inspect Clear
Console HTML CSS Script DOM
your max value is 947.83
The array values are:
732.77
755.52
805.44
714.47
761.67
947.83
805.41

Step1- Building a basic D3 chart

Right so let's get on to the interesting stuff

I'm going to assume that you are using my template and only changing the name and the code between the big markers to show you where to put your code. I am also assuming that you've run through the examples on the QlikFix so you know how to load and run extensions etc.

Setup

- Run the template.qar (did you know its just a zip with the extension renamed to .qar?) to export it to your extension folder, then just make a copy of the folder. Lets call it "01 A Basic Chart" for now but really you can call it what you want.

- Open up the Definition xml and give it this description and remove any code we don't need in this case. You can just comment lines out that we don't need, in this case pretty much everything.

```
<?xml version="1.0" encoding="utf-8"?>
<ExtensionObject Label="01 A Basic Chart" Description="First Chart Extension" PageHeight="50000">
  <!--<Dimension Label="Dimension 1" Initial="Path" TargetName="Dimension Name" />-->
    <!--<Dimension Label="Dimension 2" Initial="Measure" TargetName="Dimension Name" />-->
  <!--<Measurement Label="Measure 1" Initial="Sum(Measure)"/>-->
  <!--Text Label="Text Box 1" Type="text" Expression="TITLE"/>-->
    <!--Text Label="Checkbox 1" Type="checkbox" Initial="" />-->
    <!--Text Label="selectBox 1" Type="select" Select="select1,select2,select3" SelectLabel="selectL1,selectL2,selectL3" Expression="selectL3"/>-->
    <Initiate Name="Chart.Title" value="01 A Basic Chart" />
  <Initiate Name="Caption.Text" Value="01 A Basic Chart" />
  </ExtensionObject>
```

- Open up script.js and change the name variable at the top.

Quick Start

OK so that is the setup so let's get our hands dirty starting with a really simple chart that just displays some static figures.

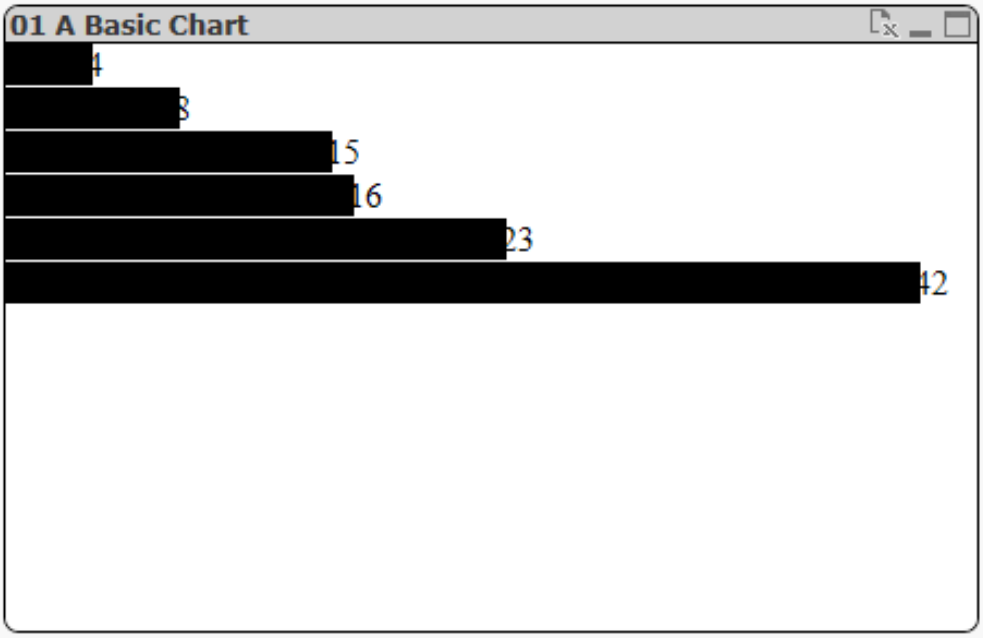
Here is the code we're doing to use to create the chart, you can just copy and paste this in if you like and we'll go through it in more detail below.

Copy and paste all of this code into the section in the template script.js which is marked at the place to put your extension code

```
_this.Element.innerHTML= '<svg class="chart"></svg>';

  var data = [4, 8, 15, 16, 23, 42];
```

```
  var width = 420,
  barHeight = 20;

  var x = d3.scale.linear()
  .domain([0, d3.max(data)])
  .range([0, width]);

  var chart = d3.select(".chart")
  .attr("width", width)
  .attr("height", barHeight * data.length);

  var bar = chart.selectAll("g")
  .data(data)
  .enter().append("g")
  .attr("transform", function(d, i) { return "translate(0," + i * barHeight
+ ")"; });

  bar.append("rect")
  .attr("width", x)
  .attr("height", barHeight-1);

  bar.append("text")
  .attr("x", function(d) { return x(d); })
  .attr("y",(barHeight-1)  / 2)
  .attr("dy", ".35em")
  .text(function(d) { return d; });
```

And that is it, if you open up a new Qvw, switch on web view and add your extension object you should have a simple if rather horrendous looking chart



Code in Detail

So to break the code down in to more manageable chunks (and remember I have no prior JavaScript training here so my explanations might be basic)

Basic Setup

You should be comfortable with the innerHTML from Barry's examples, here we're just telling it we are adding an svg with the class of chart:

```
_this.Element.innerHTML= '<svg class="chart"></svg>';
```

Here we're just adding some really basic data into an array called 'data'

```
var data = [4, 8, 15, 16, 23, 42];
```

Then just setting up a couple of display elements, so the width of the chart and the height of the bars

```
var width = 420,
barHeight = 20;
```

Set up our data range

Now we can get into the fun D3 stuff, the first thing to do is to scale our chart. D3 has a very clever way of taking a 'domain' and scaling it to a 'range' and I recommend having a read of this: [d3: scales, and color. | Jerome Cukier]

All we are doing here is telling D3 we're going to make a linear scale and then how we want to convert our domain (in this case our data variable) into the range (the size of the chart). With a bar chart it's likely we will want to start the scale at 0 so we can hard code this, and the outer is just the max of our dataset for which we can use the handy d3.max function. Then we just tell it the 'range' we want to map to, in this case hard coded at 0 again and then the outer edge is the width of our chart.

```
var x = d3.scale.linear()
.domain([0, d3.max(data)])
.range([0, width]);
```

Add a chart

The next step is to add our chart object which is very straight forward code which shouldn't need any explanation, the only point to note is that "chart" is the class we called in our svg html in the first line.

```
var chart = d3.select(".chart")
.attr("width", width)
.attr("height", barHeight * data.length);
```

Set up the bars

The next step is to create our bar data which make up the chart. When working with D3 you will see the "g" reference a lot, but this is actually svg code and to understand it and why we use "transform" then have a read here: https://www.dashingd3js.com/svg-group-element-and-d3js

What this is basically saying is
1- declare a variable bar and make it our group of data
2- For each item of data add a bar into the group
3- Set the "location" property to be starting at 0 (left) and then the index number * bar height down (From top left)

So this gives us where on the x axis the bars will start (0) and then where on the y axis they will start (depends on the bar number)

```
  var bar = chart.selectAll("g")
  .data(data)
  .enter().append("g")
  .attr("transform", function(d, i) { return "translate(0," + i * barHeight
+ ")"; });
```

You can read more about the function(d,i) in the d3 api documentation but in summary: "If `value` is a constant, then all elements are given the same attribute value; otherwise, if `value` is a function, then the function is evaluated for each selected element (in order), being passed the current datum `d` and the current index `i`, with the `this` context as the current DOM element."

Draw the bar rectangles

Then we just need to make the data visible! We do this by making the bar using a rectangle. Again this code is pretty straight forward, basically make me a rectangle with a width equivalent to our x variable (which is the 'range' we created from our 'domain') and give it a fixed height as we set earlier, but take one off the height which gives us the spacing between the bars.

```
  bar.append("rect")
  .attr("width", x)
  .attr("height", barHeight-1);
```

Add some text to the bars

The final step is to add our text to display the value, this code looks a little bit more complex but is actually very straight forward. Again we're just appending (adding) something to our bar objects, in this case some text. We then just set the attributes of the text so
1- Append our text to the bar
2- Give it an x co-ordinate, in this case we want to offset it to the end of the bar so we need to return the length of the data (x) which we set earlier.
3- Give it a y attribute which in this case is the bar height less our spacer all decided by two, which is the centre of the bar

4- Then we want to make sure that the element stays centred no matter what font size we use and also makes sure if we apply any rotation, it rotates around the centre of the text rather than the bottom, so we use the svg dy attribute to do this (just accept this works!)
5- Finally we need to tell our text what to say, which in this case is simply our data value

```
bar.append("text")
.attr("x", function(d) { return x(d); })
.attr("y",(barHeight-1)  / 2)
.attr("dy", ".35em")
.text(function(d) { return d; });
```

Wrap Up

So you can see that it is actually pretty straight forward to add a chart in to Qlikview. However in this case all we're basically doing is using Qlikview as a 'browser' to display our chart.
So what if we want to start using the power of Qlik and actually use some data and interact with the dashboard? Let's cover that off in the next couple of chapters.

Common Errors

I've found this one is normally forgetting to change the template name

References

Qlikview extension template by bmz on https://community.qlik.com/docs/DOC-3742

Extension tutorials by barryharmsen on QlikView Extension tutorials, documentation and examples - The Qlik Fix! The Qlik Fix!
Extension tips and tricks by dgudkov BI Review: Building extensions in QlikView: some hints & tips
Notepad++ https://notepad-plus-plus.org/download/v6.9.2.html
D3 Formatting http://www.jeromecukier.net/blog/2011/08/11/d3-scales-and-color/
http://www.d3noob.org/2012/12/setting-scales-domains-and-ranges-in.html
A little note on **G in svg** http://tutorials.jenkov.com/svg/g-element.html and SVG Group Element and D3.js | DashingD3js.com
D3 basic bar chart https://bost.ocks.org/mike/bar/
Qlikview SDK QlikView Version 11 SDK | Qlik Community
Basic D3 bar by ajlandry GitHub - wallyflops/d3Graph: Trying to get d3.js working in a QV extension (any graph can be added on later, in theory)
D3 API Documentation d3/API.md at master · d3/d3 · GitHub