UNIVERSITY OF WATERLOO
Faculty of Mathematics

BACKBONE.JS VS ANGULARJS

S&P Capital IQ
Toronto, Ontario

Prepared By
Gibson Wong
2B Computing and Financial Management
ID 20484836
December 19, 2014

Memorandum

To: James Wachmann
From: Gibson Wong
Date: Dec 17, 2014
Re: Work Report: Backbone.js vs AngularJS

I have prepared the enclosed report, "Backbone.js vs AngularJS," as my 2B work report for the PortfolioRisk division S&P Capital IQ. This report, the third of four work reports has not received an academic credit and is required by the Co-operative Education Program as part of my Bachelor of Computing and Financial Management Honors degree requirements.

As a software developer, my main duty is to manage the front-end client side of the PortfolioRisk web application. This includes implementing new features, maintaining old code as well as fixing bugs. In order to fulfill my tasks, I rely on tools like JavaScript and frameworks like Backbone.js. This report focuses on the differences between the two frameworks, Backbone.js and AngularJS, and determines which framework is most suitable for the company.

The Faculty of Mathematics requests that you evaluate this report for command of topic and technical content/analysis. Following your assessment, the report, together with your evaluation, will be submitted to the Math Undergrad Office for evaluation on campus by qualified work report markers. The combined marks determine whether the report will receive credit and whether it will be considered for an award.

I would like to thank you for your assistance in preparing this report.

TABLE OF CONTENTS

LIST OF FIGURES

EXECUTIVE SUMMARY

As the popularity of web application increases, a wide variety of frameworks began to emerge. The purpose of a framework is to provide developer with functionality to properly organize and structure code. Currently, the PortfolioRisk web application uses Backbone.js as their client-side framework. Backbone.js is a light framework that follows a variation of the Model-View-Controller (MVC) pattern. This pattern allows the model and view to communicate with each other through the use of controllers. As a result, this framework provides a method to help developers to properly structure their solution. The framework also gives developers the freedom to easily create and wire up web components together.

Similarly, AngularJS is a framework that also implements the MVC pattern. The model and view is synced automatically through a through a process called two-way data-binding. Another major benefit of using AngularJS is the ease in writing test cases for the project. This gives developers the confidence to implement changes without creating new bugs.

Despite the major benefits of using AngularJS, the final recommendation is to continue with Backbone.js. This is because the company is currently busy with balancing adding new features as well as proper code maintenance. Migrating frameworks right now is extremely risky and time consuming. However, once the company becomes more stable, extra thought should be put in place in considering AngularJS for the client-side framework.

1.0 INTRODUCTION

The growing demand for dynamic web applications has driven an increase in new JavaScript web framework. The most popular type of framework follows an architect pattern called Model-View-Controller (MVC). This pattern addresses the issue of separation of concerns by splitting up the problem into three components: the model which contains the data, the view which contains the display and the controller which glues the model and view together. As this architecture helps solve various problems, many frameworks are beginning to implement the MVC pattern.

The increase in different frameworks has also led to a popular, yet never ending debate in finding the best framework. Although there is no concrete answer, comparing the different frameworks available will yield important findings. This report will compare the two popular frameworks, Backbone.js and AngularJS; the former of which is currently used by S&P Capital IQ. Firstly, a broad overview of the two different frameworks will be provided. The benefits of switching from Backbone.js to AngularJS will then be closely analyzed. Next, any drawbacks that may occur from the migration will be investigated and possible solutions will be revealed. Finally, a recommendation for S&P Capital IQ will be provided.

2.0 ANALYSIS

As previously mentioned, many web frameworks currently are built using the MVC pattern. In essence, each framework has their own idea on how to organize models, views and controllers.

These different ideas are the key criteria in creating a unique framework and are one of the main reasons behind the intense debates found online.

Both Backbone.js and AngularJS have been around for the same amount of time. However up until the beginning of 2013, Backbone.js was the more popular weapon of choice. As Figure 1 shows, the year 2013 marks the cross over between interests in frameworks, and is caused by the significant increase for interest in AngularJS (blue colored line). On the other hand, interest for Backbone.js (red colored line) has slowly decreased thereafter.

Why did major shift in interest occur and should the company follow this trend?

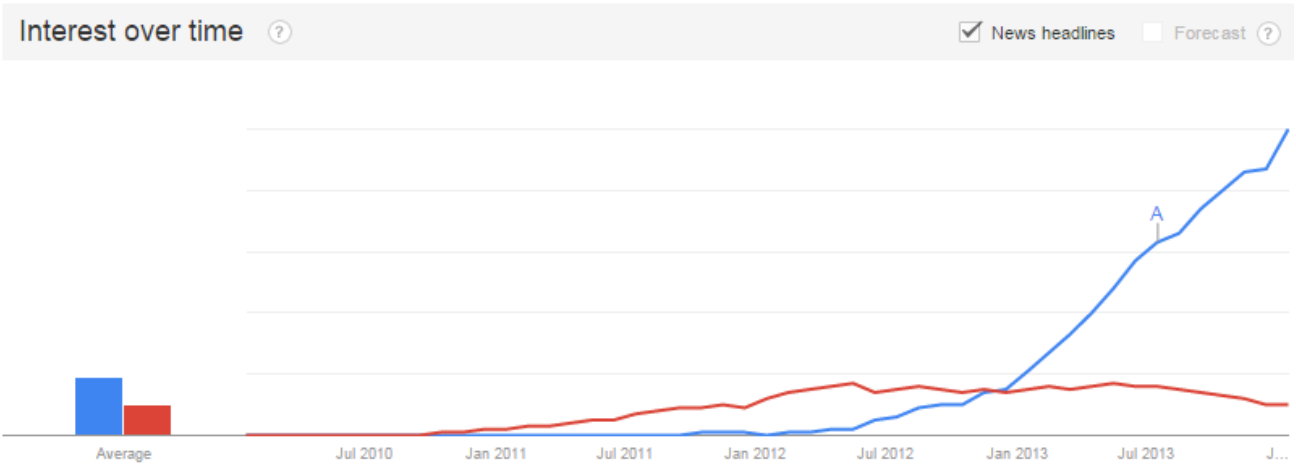

Figure 1 – Interest Between Frameworks Over Time.

2.1 Backbone.js

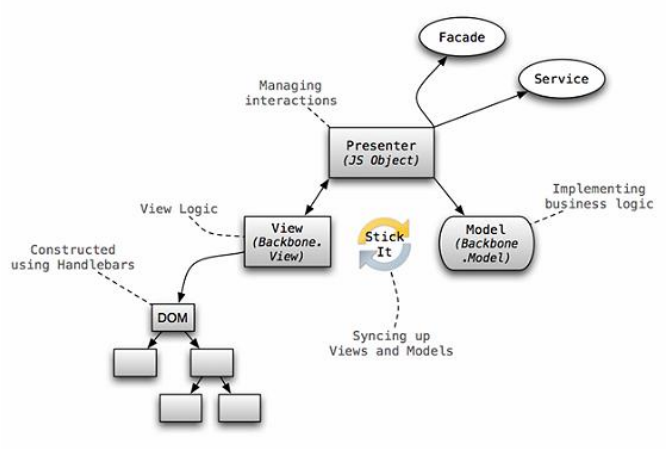

Figure 2 – The Backbone.js Pattern.

To start off, Backbone.js is a small JavaScript framework that provides the basic tools to structure the MVC pattern. It relies on the Underscore library and often times, the jQuery library in order provide a complete platform for building web applications. Despite being the small framework that it is, Backbone.js offers a good set of conventions to help structure the code properly. The diagram above shows how code is typically structured through the Backbone.js pattern. This is done through the event-driven communication between Backbone's model and views. One can attach an event listener to an attribute from the model and reflect the changes onto the view. Because of the lack in Backbone controllers (since Backbone views act as both views and controller), many programmers recognize Backbone.js as a framework that follows the Model-View-ViewModel (MVVM) pattern. Regardless of the title of the pattern, Backbone.js provides a respectable solution to organizing large and complex JavaScript code.

Another core feature of Backbone.js is that it is a small and minimalistic framework. This allows for a lower learning curve on the user and at the same time, provides clean and organized code. Furthermore, developers have the freedom to create and wire up components to their liking. If there are problems that cannot be solved with the Backbone framework alone, programmers can

easily create their own plug-ins or include a third party library. That being the case, developers are often required to create the architecture and wiring themselves, which in the end, is time consuming and requires good attention to writing clean code.

2.2 AngularJS

On the other hand, AngularJS is a larger framework that was built to relief the issue of tightly coupled code between declarative programming with imperative programming. To understand the differences, imperative programming can be thought as "how to do something, and what you want to happen" while declarative programming can be thought as "what you would like to happen". In Backbone.js, both of these programming styles are incorporated in the Backbone view. As a result there is a tight coupling between the DOM manipulation (manipulation of web elements) and the business logic. AngularJS solves this issue by enforcing declarative programming into angular directives and imperative programming into angular controllers. Thus, there exists a clear separation between the presentation and the data. Furthermore, AngularJS was created to encourage developers to follow a test-driven development process. In other words, developers should always think of different test cases first, before diving deep into writing code. The diagram on the illustrates the

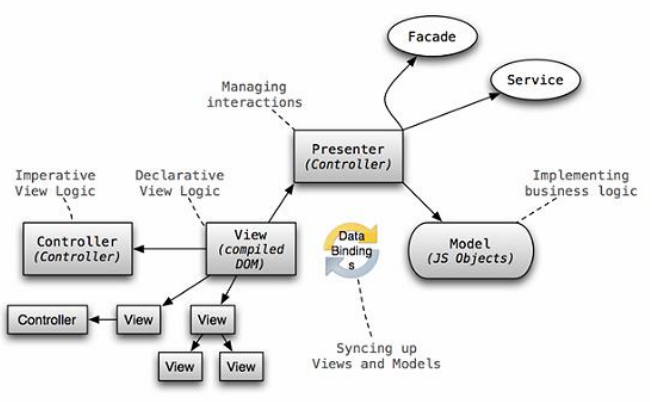

Figure 3 – The AngularJS Pattern

pattern used for AngularJS

Unlike Backbone.js, AngularJS is extremely opinionated and requires developers to follow Angular's way of development. Many programming practices derived from Backbone.js cannot be applied for development with AngularJS. For instance, creating user interface with JavaScript (often done by Backbone.js developers) is frowned upon when using AngularJS. As a result, a much steeper learning curve is required.

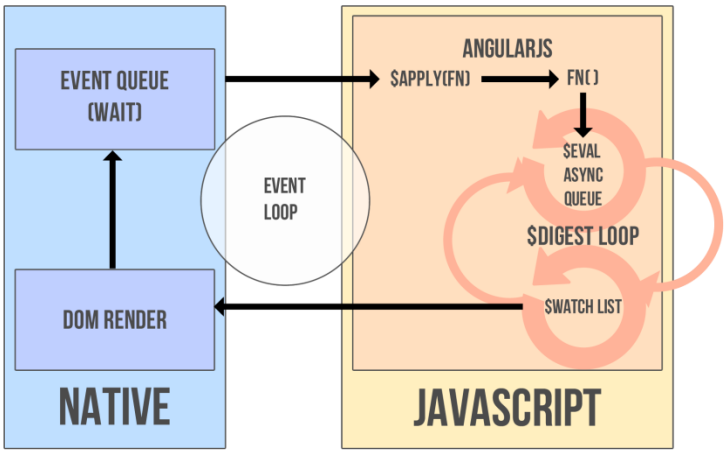

Figure 4 – AngularJS' Event Cycle

The syncing between models and views are also quite different when comparing Backbone.js to AngularJS. Instead of listening to changes from the model and the view, AngularJS has a built in data-binding system. AngularJS does this by extending the regular web flow by adding its own event loop, called the digest loop. It then searches for all bindings in the code and apply them into a watch list. The bindings are resolved (using a method called dirt checking) when the digest loop runs and any changes found are updated in the respective model or view. Since changes from the model can update the view, and vice versa, this method of data-binding is called two-way data-binding.

2.3 Advantages of Switching Frameworks

There are numerous advantages that can be gained by switching from a Backbone.js framework to an AngularJS framework. To begin with, AngularJS includes a large application program interface (API) that can be used to help simplify common tasks of a web application. For instance, loading a list of data and filtering them in real time can be done in one line (using Angular directives like ng-repeat and ng-filter). Being able to filter data in real time is also an impressive feature and is easily created by using Angular's two-way data-binding system. In order to do the same thing, Backbone developers will need to attach jQuery event to input boxes as well listeners to the model to listen for changes. This entire process is quite tedious since it is required each time a developer wants to sync up the model and the view.

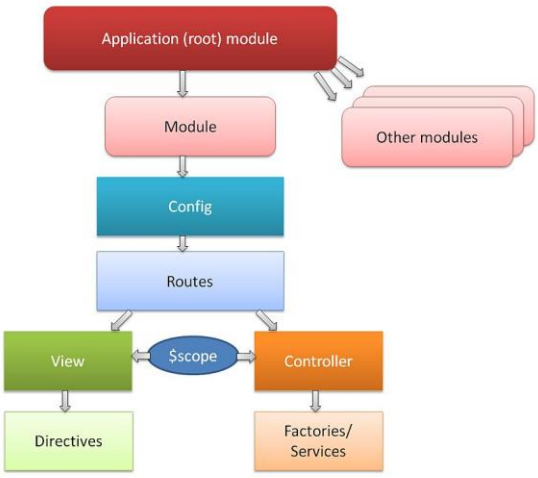

Figure 5 – AngularJS Components

Furthermore, the overall structure using AngularJS is reasonably clean and readable. This is because each component can easily be fit in the following types: Views (templates), Controllers, Factories, Services and Filters. Each of these can then be tested individually and re-used whenever necessary. The diagram on the left illustrates AngularJS' different components.

On the other hand, Backbone.js only provides View, Model and Collections (similar to Models) and also relies on Underscore's templating engine to create the templates. Most of the logic are focused heavily on Backbone Views, and as a result, hinders the ability to re-use code.

Finally, the most important benefit that can be gained from AngularJS is that it enforces developer to write test-driven code. The ability to test individual modules (this process is called Unit-Testing) allows developers to gain confidence whenever editing existing code. This is especially useful in the case for large-scale project as many developers will be making changes to the project regularly. The ease in unit testing in AngularJS is created by a method called dependency injection. Each unit in the project is highly decoupled, separate from other modules and can be tested individually. In the event that modules depend on one another, testing can be done simply by injecting modules together.

2.4 Overcoming the Difficulties

Although there are many advantages of AngularJS, switching frameworks can also yield some difficulties. As previously stated, AngularJS is a framework with a steeper learning curve. Switching from Backbone.js to AngularJS requires each developer to learn the new frameworks and adapt to the new programming styles. That being said, the growing community for AngularJS is a major asset for developers since solutions to problem can easily be found online. However, once developers are familiar with this framework, an increase in productivity can be seen. Since there is a lot of built in functionality, developers can save time writing tedious boilerplate code (commonly used function) and spend more time creating features. The amount

of time spent fixing bug will also be reduced as each module will be tested before making changes to the project.

As the programming practices between the two frameworks are quite different, migrating code will also take a decent amount of time. Furthermore, things that are originally done in Backbone.js may not be the same as AngularJS. For instance, Backbone.js prefers rendering template and nest views to create the UI. AngularJS does not have a concrete way of doing this. Instead, this framework prefers to use one single view and is rendered automatically on Angular's bootstrapping process. That being said, it is not impossible to nest views in AngularJS. The solution to this problem is to create angular directives. These directives return objects that can hold templates as well as added functionalities. As a result, we can create one view and have nested directives which act similarly to Backbone's nested views. Furthermore, this solution follows well with AngularJS' coding practice to allow each unit (in this case, directives) to be easily tested. To see a concrete example, view Appendix A.

Another major difference between coding practices of the two frameworks is the ability to integrate third party library. Since Backbone.js is a small and flexible framework, integrating third party library is very easy. For example, S&P Capital IQ relies on libraries like jQuery, slickgrid, Moment.js and many more. On the other hand, using third party library in AngularJS is much more difficult. These third party libraries will need to be converted into angular directives or services while keeping testability in mind. Furthermore the amount of third-party library for AngularJS is much smaller than Backbone.js. However, this is not entirely a major concern as

AngularJS is capable of replacing many libraries commonly used in Backbone.js. For instance, many functionallies from the jQuery library are already a built in feature of AngularJS' animate directive. An example of incorporating jQuery into Angular can be found in Appendix B

3.0 CONCLUSION

With so many frameworks available, web applications are evolving to become more interactive and more dynamic. Given each framework is distinct from one another, analyzing the benefits and downfall each framework comes with is always extremely important. Backbone.js is a lightweight framework giving developers the basic tools to create good structure and stable foundation for clean and organize code. It furthermore allows developers the freedom to build upon the framework through a wide variety of third party libraries. As a result, small applications can be easily whipped up and maintained to your own liking. AnuglarJS, on the other hand, limits developer's programming choices to follow the conventions provided by the framework. With the added ability of two-way data-binding, developers no long need to manually sync the model and view together. By forcing developers to split up large problems into decoupled and maintainable code, testing solutions are easier than ever.

As the intense debates for the greatest framework continue, one can only question if an answer will ever exist. Instead of blindly following the trend, developers should pool in time to carefully research and analyze the frameworks that are best suited for situation. Each framework will have its own benefits and downfalls. The ability to adapt to these changes is the key in finding the most appropriate framework for the project.

4.0 RECOMMENDATION

The PortfolioRisk division of S&P Capital IQ is currently using Backbone.js as its main front-end framework. Using Backbone, the team is able to build the product and implement features extremely quickly. As the project continues to expand, maintaining the solution has become a major issue. Constant time and effort is set in place to fix bugs as well as re-factor code to increase quality. Using AngularJS will be able to solve these problems as it enforces developers to constantly test their code. In addition, implement features is much easier since two-way data-binding automatically syncs up models and view.

Although implementing AngularJS for the front-end framework will be extremely beneficial to the company, I currently would not recommend this change. Instead, I would recommend further research and proper training before diving into to the migration. Since the current team is focused on juggling between bug fixes, code re-factoring and features implementation at the same time, migrating frameworks right now can be quite dangerous. Only when the company has decided to slow down on pushing out features, should the team considering switching frameworks.

REFERNECES

(n.d.). Retrieved December 17, 2014, from http://jan.varwig.org/archive/angularjs-views-vs-directiveshttps://github.com/angular/angular.js/wiki/Understanding-Scopes#JSproto

"Thinking in AngularJS" if I have a jQuery background? (n.d.). Retrieved December 17, 2014, from http://stackoverflow.com/questions/14994391/thinking-in-angularjs-if-i-have-a-jquery-background

Albretch, A. (n.d.). Adam Albretch. Retrieved December 17, 2014, from http://adamalbrecht.com/2013/12/12/creating-a-simple-modal-dialog-directive-in-angular-js/

Contrasting Backbone and Angular. (n.d.). Retrieved December 17, 2014, from http://www.infoq.com/articles/backbone-vs-angular

Directive: Link vs compile vs controller. (n.d.). Retrieved December 17, 2014, from http://stackoverflow.com/questions/15676614/directive-link-vs-compile-vs-controller

Hooks, J. (n.d.). Let's Make Full-Ass AngularJS Directives. Retrieved December 17, 2014, from http://joelhooks.com/blog/2014/02/11/lets-make-full-ass-angularjs-directives/

Koren, I. (n.d.). My Own Angle on AngularJS - Pros and Cons. Retrieved December 17, 2014, from https://connect.liveperson.com/community/developers/blog/2014/02/04/my-own-angle-on-angularjs--pros-and-cons

Lerner, A. (2013). The Digest Loop and $apply. In *Ng-book: The complete book on AngularJS*. S.I.: Fullstack.io :.

Next Big Thing. (n.d.). Retrieved December 17, 2014, from http://blog.nebithi.com/backbone-and-angular-demystifying-the-myths/

Orlenko, V. (2013, March 6). AngularJS for jQuery Developers. Retrieved December 17, 2014,

from http://www.artandlogic.com/blog/2013/03/angularjs-for-jquery-developers/

Porath, M. (n.d.). Quora. Retrieved December 17, 2014, from http://www.quora.com/What-are-the-advantages-of-Backbone-js

Shaked, U. (n.d.). AngularJS vs. Backbone.js vs. Ember.js. Retrieved December 17, 2014, from https://www.airpair.com/js/javascript-framework-comparison

Test your JavaScript, CSS, HTML or CoffeeScript online with JSFiddle code editor. (n.d.). Retrieved December 17, 2014, from http://jsfiddle.net/theparabolink/dnCHA/

Varwig, J. (n.d.). Jan Varwig — Somewhere between Hello World and HAL9000. Retrieved December 17, 2014, from http://jan.varwig.org/archive/angularjs-views-vs-directives

APA formatting by BibMe.org.

Appendix A - Nesting Views With AngularJS

Using HTML as templating engines, the following are the individual views used for nesting

```
<!--Method-property.html-->
<div>
    <h4>Instrument Dimension</h4>
    <instrument-selection></instrument-selection>
    <br>
    <pricing-parameter></pricing-parameter>
    <hr>
    <h4>Scenario Dimension</h4>
    <scenario-dimension></scenario-dimension>
</div>

<!--Instrument-selection.html-->
<div>
    <span style="display:block">Instrument Selection</span>
    <div style="float:left">
        *Portfolio
        <input type="text" ng-model="methodProperty.instrument.portfolio">
    </div>
    <div style="margin-left: 459px;">
        As Of Date
        <input type="text" ng-model= "methodProperty.instrument.asOfDate">
        <button type="button">  </button>
    </div>
</div>

<!--Pricing-parameter.html-->
<div>
    <span style="display:block">Pricing Parameters</span>
    <div style="float:left">
        *Instrument Model
        <input type="text" ng-model="methodProperty.pricing.IM">
    </div>
    <div style="margin-left: 459px;">
        Pricing Date
        <input type="text" ng-model="methodProperty.pricing.priceDate">
        <button type="button">  </button>
    </div>
    <div>
        *Reporting Currency
        <input type="text" ng-model="methodProperty.pricing.reportCurrency">
    </div>
</div>
```

```
<!--Scenario-dimension.html-->
<div>
    <div style="float:left">
        *Scenario Set
        <input type="text" ng-model="methodProperty.scenario.scenSet">
    </div>
        <div style="margin-left: 459px;">
        *Metrics
        <input type="text" ng-model="methodProperty.scenario.metrics">
        <button type="button">  </button>
    </div>
    <div>
        Baseline
        <input type="text" ng-model="methodProperty.scenario.baseline">
        <button type="button">  </button>
    </div>
</div>
```

Unless extra functionalities are needed, the following is the code to define the instrumentSelection directive. Other directives are defined similarly.

```
'use strict';

angular.module ('myApp.view1.method-property.instrument-selection-directive', [])

.directive('instrumentSelection', function () {
    return {
        restrict: 'E',
        templateUrl: 'view1/components/method-property/InstrumentDimension/instrument-selection.html',
    }
});
```

Inject all the directives together in the parent module.

```
'use strict';

angular.module('myApp.view1.method-property', ['myApp.view1.method-property-directive',
                                                'myApp.view1.method-property.instrument-selection-directive',
                                                'myApp.view1.method-property.pricing-parameter-directive',
                                                'myApp.view1.method-property.pricing-parameter.nest-directive',
                                                'myApp.view1.method-property.scenario-dimension-directive'])


.controller('MethodCtrl', ['$scope', function($scope) {
}]);
```

Appendix B – Example of jQuery Dialogs Incorporated Into AngularJS

Code for creating the r2Dialog directive

```
.directive('r2Dialog', function ($timeout) {
    return {
        restrict: 'E',
        replace: false,
        transclude: true,
        template: '<div ng-transclude></div>',
        scope: {
            okButton: '@',
            okCallback: '=',
            cancelButton: '@',
            cancelCallback: '=',
            open: '@',
            title: '@',
            width: '@',
            height: '@',
            autoOpen: '@',
            resizable: '@',
            closeOnEscape: '@',
            hideCloseButton: '@',
            enableClosButton: '@',
            modal: '@',
            draggable: '@',
            buttons: '@',
            setClose: '='
        },
        controller: function ($scope, $element) {
            var hideCloseButton = $scope.hideCloseButton || true;
            ++dialogNumber;
            var dialogClass = 'r2-dialog-' + dialogNumber;

            $scope.dialogOptions = {
                autoOpen: $scope.autoOpen||false,
                modal: $scope.modal || true,
                resizable: $scope.resizable || false,
                draggable: $scope.draggable || true,
                dialogClass: dialogClass,
                height: $scope.height ||'auto',
                width: $scope.width ||'auto',
                closeOnEscape: $scope.closeOnEscape || false,
                close: function () {
                    console.log('closing...');
                },
                open: function (event, ui) {
                    if(hideCloseButton == true) {
                        $(".ui-dialog-titlebar-close", ui.dialog).hide();
                    }
                }
            };
            $scope.dialogOptions['buttons'] = [];
```

```
54          },
55          link: function (scope, element, attrs, ctrl) {
56              if(attrs.okButton) {
57                  var btnOptions = {
58                      text: attrs.okButton,
59                      click: function() {
60                          scope.$apply(scope.okCallback);
61                      }
62                  };
63                  scope.dialogOptions['buttons'].push(btnOptions);
64              }
65
66              if(attrs.cancelButton) {
67                  var btnOptions = {
68                      text: attrs.cancelButton,
69                      click: function() {
70                          scope.$apply(scope.cancelCallback());
71                      }
72                  };
73                  scope.dialogOptions['buttons'].push(btnOptions);
74              }
75              // Initialize the element as a dialog
76              // For some reason this timeout is required, otherwise it doesn't work
77              // for more than one dialog
78              $timeout(function() {
79                  $(element).dialog(scope.dialogOptions);
80              },0);
81              // This works when observing an interpolated attribute
82              // e.g {{dialogOpen}}.  In this case the val is always a string and so
83              // must be compared with the string 'true' and not a boolean
84              // using open: '@' and open="{{dialogOpen}}"
85              attrs.$observe('open', function(val) {
86                  console.log('observing open val=' + val);
87                  if (val == 'true') {
88                      console.log('open');
89                      $(element).dialog("open");
90                  }else  {
91                      console.log('close');
92                      $(element).dialog("close");
93                  }
94              });
95              // This allows title to be bound
96              attrs.$observe('title', function(val) {
97                  console.log('observing title: val=' + val);
98                  $(element).dialog("option", "title", val);
99              });
100         }
101     }
102 });
```

Code for the view (template)

```
<button  ng-click="openDialog('deleteButton')" style="float:right; margin-right:20px;">Delete Simulation</button>
<r2-dialog
    title="Delete Simulation"
    open="{{deleteButton.isOpen}}"
    ok-button="OK"
    ok-callback="deleteButton.onOkCallback"
    cancel-button="Cancel"
    cancel-callback="deleteButton.onCloseCallback"
    width="300">
    Message: ({{deleteButton.deleteSimDialogMessage}})
</r2dialog>
```

Code for the data in the controller

```
$scope.deleteButton = { isOpen : false,
                        deleteSimDialogMessage : "Are you sure you wish to delete this simulation?",
                        onOkCallback: function () {
                            alert ("ok callback");
                            $scope.deleteButton.isOpen = false;
                        },
                        onCloseCallback: function() {
                            alert ("Close callback");
                            $scope.deleteButton.isOpen  = false;
                        }}
$scope.openDialog = function (isDialogOpenControl) {
    $scope[isDialogOpenControl].isOpen = true;
}
```