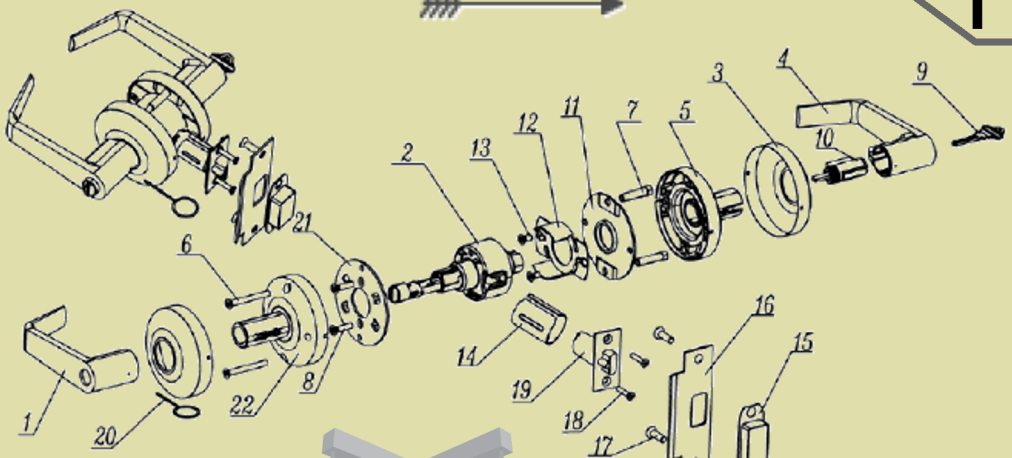
Learn to use javascript AMD in your apps the easy way

Structuring BACKBONE.JS Code with REQUIRE.JS RequireJS and Marionette.js Modules



A Gentle Introduction

by David Sulc

Structuring Backbone Code with RequireJS and Marionette Modules

Learn to use javascript AMD in your apps the easy way

David Sulc

This book is for sale at http://leanpub.com/structuring-backbone-with-requirejs-and-marionette

This version was published on 2015-09-20

Tweet This Book!

Please help David Sulc by spreading the word about this book on Twitter!

The suggested tweet for this book is:

I just bought @davidsulc's book on @marionettejs and RequireJS. Interested? Check it out at https://leanpub.com/structuring-backbone-with-requirejs-and-marionette

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#

Also By David Sulc

Backbone.Marionette.js: A Gentle Introduction

Backbone.Marionette.js: A Serious Progression

Marionette.js: Testing and Refactoring

Contents

Work in Progress

This book is currently being written, and I'd love to hear from you! The ultimate goal, of course, is to cover the sticking points readers have trouble understanding or with using RequireJS and Marionette, so you'll be comfortable using RequireJS in your own applications.

Who This Book is For

This book is first and foremost for the readers of "Backbone.Marionette.js: A Gentle Introduction[1]" who want to take their knowledge to the next level, by using RequireJS to structure their code and load dependencies.

To follow along, you'll need to have at least passing familiarity with the Contact Manager application developed in my previous book. Put another way, the focus of this book is how to use RequireJS with Marionette: it does NOT explain how to leverage the various components Marionette provides (that's covered in the book linked above).

[1] https://leanpub.com/marionette-gentle-introduction

Following Along with Git

This book is a step by step guide to rebuilding the Contact Manager application to use RequireJS. As such, it's accompanied by source code in a Git repository hosted at https://github.com/davidsulc/structuring-backbone-with-requirejs-and-marionette[2].

Throughout the book, as we code our app, we'll refer to commit references within the git repository like this:

Git commit with our basic index.html:

dcc76a70bd5520add1d4ebdc42320aacb0c0d77f[3]

This will allow you to follow along and see exactly how the codebase has changed: you can either look at that particular commit in your local copy of the git repository, or click on the link to see an online display of the code differences.

Any change in the code will affect all the following commit references, so the links in your version of the book might become desynchronized. If that's the case, make sure you update your copy of the book to get the new links. At any time, you can also see the full list of commits here[4], which should enable you to locate the commit you're looking for (the commit names match their descriptions in the book).

Even if you haven't used Git yet, you should be able to get up and running quite easily using online resources such as the Git Book[5]. This chapter is by no means a comprehensive introduction to Git, but the following should get you started:

- Set up Git with Github's instructions[6]
- To get a copy of the source code repository on your computer, open a command line and run

```
git clone git://github.com/davidsulc/structuring-backbone-with-requirejs-and-ma\
rionette.git
```

- From the command line move into the `structuring-backbone-with-requirejs-and-marionette` folder that Git created in the step above, and execute

[2] https://github.com/davidsulc/structuring-backbone-with-requirejs-and-marionette
[3] https://github.com/davidsulc/structuring-backbone-with-requirejs-and-marionette/commit/dcc76a70bd5520add1d4ebdc42320aacb0c0d77f
[4] https://github.com/davidsulc/marionette-gentle-introduction/commits/master
[5] http://git-scm.com/book
[6] https://help.github.com/articles/set-up-git

```
git show dcc76a70bd5520add1d4ebdc42320aacb0c0d77f
```

to show the code differences implemented by that commit:

- '-' lines were removed
- '+' lines were added

You can also use Git to view the code at different stages as it evolves within the book:

- To extract the code as it was during a given commit, execute

  ```
  git checkout dcc76a70bd5520add1d4ebdc42320aacb0c0d77f
  ```

- Look around in the files, they'll be in the exact state they were in at that point in time within the book
- Once you're done looking around and wish to go back to the current state of the codebase, run

  ```
  git checkout master
  ```

What if I don't want to use Git, and only want the latest version of the code?

You can download a zipped copy of the repository[7]. This will contain the full Git commit history, in case you change your mind about following along.

Jumping in for Advanced Readers

My goal with this book is to get you comfortable enough to use RequireJS in your own Marionette projects, and it rebuild an existing application from start to finish with RequireJS. Therefore, after presenting the main concepts you need to know when using RequireJS, the book guides you in rewriting the ContactManager application by pointing out various aspects to bear in mind, and then providing a step by step guide to the actual implementation.

Although you'll learn the most by following along with the code, you can simply skim the content and checkout the Git commit corresponding to the point in the book where you wish to join in.

[7] https://github.com/davidsulc/structuring-backbone-with-requirejs-and-marionette/archive/master.zip

Why Use AMD?

Asynchronous Module Definition (or AMD for short) is one answer to the problems that plague modern web development:

- web applications require lots of javascript code;
- to manage application complexity, javascript code needs to be broken down into smaller files;
- these smaller javascript files have dependencies on one another that aren't clear from their inclusion via `<script>` tags: the tag sequence only gives a rough sense of order, not a dependency tree;
- loading a long list of javascript files with includes slows down your web page: each file requires a separate HTTP request, and is blocking.

By using RequireJS in a web app, you get the following benefits:

- your javascript code can be distributed among many smaller files;
- you explicitly declare the dependencies each javascript module requires (both libraries and other modules);
- you can build a single, optimized, and minified javascript file for production deployment.

Getting Started

This book uses Marionette 2.3.2. If you wish to learn an earlier version of Marionette (e.g. you've inherited a project with an older version), refer to the older book version included as a zip. The code using Marionette 1.7.4 is available on Github in the `marionnette-pre-v2` branch[8].

First, let's grab a copy of the source code for our basic Contact Manager application (as it was developed in my "Backbone.Marionette.js: A Gentle Introduction[9]" book) from here[10]. We're doing this for convenience, since we'll now have all libraries and assets (images, icons, etc.) available and won't have to deal with fetching them in the next chapters. Downloading the entire app also means we have all of our modules ready to go, simply waiting to be integrated in our RequireJS app.

Now, let's start with a basic `index.html` file, by removing all javascript includes, and templates:

Our initial index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Marionette Contact Manager</title>
    <link href="./assets/css/bootstrap.css" rel="stylesheet">
    <link href="./assets/css/application.css" rel="stylesheet">
    <link href="./assets/css/jquery-ui-1.10.3.custom.css" rel="stylesheet">
  </head>

  <body>
    <div id="app-container">
      <div id="header-region"></div>
      <div id="main-region" class="container">
        <p>Here is static content in the web page. You'll notice that it
           gets replaced by our app as soon as we start it.</p>
      </div>

      <div id="dialog-region"></div>
```

[8] https://github.com/davidsulc/structuring-backbone-with-requirejs-and-marionette/tree/marionette-pre-v2

[9] https://leanpub.com/marionette-gentle-introduction

[10] https://github.com/davidsulc/structuring-backbone-with-requirejs-and-marionette/archive/dcc76a70bd5520add1d4ebdc42320aacb0c0d77f.zip

```
20      </div>
21    </body>
22  </html>
```

You may have noticed that we've got some CSS includes that aren't needed yet. We'll just leave them there, so we can avoid dealing with them later: they'll already be included when we do need them.

Git commit with our basic index.html:

2cf4f77cbffaab86aaf9b355410a6d2f46f58d29[11]

Adding RequireJS to the Mix

With our basic `index.html` in place, let's move on to our next goal: using RequireJS to load our libraries. So let's get RequireJS from here[12] and save it in `assets/js/vendor/require.js`. Now, we can add RequireJS to our `index.html` (line 22):

Adding RequireJS (index.html)

```
1   <!DOCTYPE html>
2   <html lang="en">
3     <head>
4       <meta charset="utf-8">
5       <title>Marionette Contact Manager</title>
6       <link href="./assets/css/bootstrap.css" rel="stylesheet">
7       <link href="./assets/css/application.css" rel="stylesheet">
8       <link href="./assets/css/jquery-ui-1.10.3.custom.css" rel="stylesheet">
9     </head>
10
11    <body>
12      <div id="app-container">
13        <div id="header-region"></div>
14        <div id="main-region" class="container">
15          <p>Here is static content in the web page. You'll notice that it
16             gets replaced by our app as soon as we start it.</p>
17        </div>
18
19        <div id="dialog-region"></div>
```

[11] https://github.com/davidsulc/structuring-backbone-with-requirejs-and-marionette/commit/2cf4f77cbffaab86aaf9b355410a6d2f46f58d29

[12] http://requirejs.org/docs/release/2.1.8/comments/require.js

```
20      </div>
21
22      <script src="./assets/js/vendor/require.js"></script>
23    </body>
24  </html>
```

So how do we actually load our libraries with RequireJS? We're going to add a `data-main` attribute to the `script` tag, which will make RequireJS automatically load the provided javascript file. This file, in turn, will configure RequireJS per our environment, load the top-level dependencies and libraries, and then start our Contact Manager application. So let's change our `script` tag on line 22 to

```
<script data-main="./assets/js/require_main.js"
          src="./assets/js/vendor/require.js"></script>
```

That was easy enough! Of course, we now need to actually write *require_main.js* to provide configuration directives. Let's start by simply using RequireJS to load jQuery and display jQuery's version number:

assets/js/require_main.js

```
1  require(["vendor/jquery"], function(){
2    console.log("jQuery version: ", $.fn.jquery);
3  });
```

If you refresh `index.html`, you'll now see jQuery's version number printed in the console. How is this happening? First, RequireJS determines the `baseUrl`, which is the path from which all other paths are computed. By default, it is set to the path provided to the `data-main` attribute above (minus the file portion, of course). So in our case, the `baseUrl` is *assets/js*.

Then, within *require_main.js*, we call RequireJS's `require` function, which takes 2 arguments:

1. an array of dependencies that need to be loaded before executing the callback function;
2. a callback function that gets executed once all the required dependencies have been loaded.

Note that RequireJS also defines a `requirejs` function, which is interchangeable with `require`.

In the dependency array, we provide the path to the file we want to load *relative to the `baseUrl` path*. As we indicated jQuery was located at *vendor/jquery*, and our `baseUrl` is *assets/js* (because that's

where our *require_main.js* file is located), RequireJS will look for the library file in *assets/js/vendor/jquery.js*. So our code essentially instructs "once jQuery is loaded from *assets/js/vendor/jquery.js*, print its version number."

Note that we specify file without extensions. For example, to require jQuery we had "vendor/jquery" as the dependency, and RequireJS automatically added the ".js" extension.

But we can still improve our code by adding some global configuration:

assets/js/require_main.js

```
1  requirejs.config({
2    baseUrl: "assets/js",
3    paths: {
4      jquery: "vendor/jquery"
5    }
6  });
7
8  require(["jquery"], function(jq){
9    console.log("jQuery version: ", jq.fn.jquery);
10   console.log("jQuery version: ", $.fn.jquery);
11 });
```

Due to the equivalence of `require` and `requirejs`, the config method could also be called via `require.config`. Why do I use `requirejs` on line 1 and then `require` on line 8? It's just a matter of personal taste: I find that `requirejs.config` explicitly indicates we are configuring RequireJS, and using the `require` function call reads better (i.e. "require these files before proceeding"). But be aware of the `requirejs` and `require` synonyms when you look at RequireJS code.

So we've added some configuration, but what does it do? First, we provide a `baseUrl` path. This isn't really necessary in our case (since we're using the same path as the default), but I find it helps with clarity: since all the other paths indicated in the file will be defined relative to the `baseUrl`, it's practical to have it explicitly defined on the same page.

Next, we've added a `paths`[13] object to specify paths that aren't directly under the `baseUrl` path. This allows us to create a "jquery" key to use when requiring the library, and RequireJS will know it should be loaded from the *vendor/jquery* location.

[13] http://requirejs.org/docs/api.html#config-paths

From the documentation[14]:

The path that is used for a module name should **not** include an extension, since the path mapping could be for a directory. The path mapping code will automatically add the .js extension when mapping the module name to a path.

By using path fallbacks[15], you can easily load libraries from CDNs and default to using a local version should the CDN be unavailable.

With this configuration in place, we can now simply require "jquery" and RequireJS will know where to look. In addition, note that RequireJS will provide the array of dependencies as arguments to the callback function: in the code above, we have the "jquery" dependency assigned to the `jq` variable, which allows us to then call `jq.fn.jquery` to get jQuery's version number. As you can tell from the console output, using the provided callback argument is equivalent to using the global `$` variable.

JQuery's case is a bit special, so let's talk about it a bit. Ever since version 1.7, jQuery is AMD-compatible meaning that it can be loaded with a module loader such as RequireJS. In that case, the module loader will receive a reference to jQuery which can be used in the callback function. All of this happens *in addition* to jQuery's registering the global `$` variable.

However, with most other modules and libraries, no global variables will be registered. That's one of the objectives when using RequireJS: don't register objects in the global namespace, and require the ones you need instead, passing them into the callback function.

Using Shims

We've seen how to load an AMD-compatible library (namely, jQuery), so let's move on to loading one that is *not* compatible with AMD: Underscore. By using the new `shim` object in RequireJS, we can still load AMD-incompatible scripts with RequireJS: all we need to do is provide a little more information. Here we go:

[14] http://requirejs.org/docs/api.html#config-paths
[15] http://requirejs.org/docs/api.html#pathsfallbacks

assets/js/require_main.js

```
1  requirejs.config({
2    baseUrl: "assets/js",
3    paths: {
4      jquery: "vendor/jquery",
5      underscore: "vendor/underscore"
6    },
7
8    shim: {
9      underscore: {
10       exports: "_"
11     }
12   }
13 });
14
15 require(["underscore", "jquery"], function(un){
16   console.log("jQuery version: ", $.fn.jquery);
17   console.log("underscore identity call: ", _.identity(5));
18   console.log("underscore identity call: ", un.identity(5));
19 });
```

First, we indicate where Underscore is located (line 5). With that done, we add the "underscore" key to the `shim` object and indicate that it should return the value of the "_" global to the callback function. Notice that, thanks to the shim, we have access to both the global "_" value (line 17) and the `un` callback value (line 18). If we hadn't used a shim, we wouldn't be able to use the `un` callback value, only the "_" global.

Since writing this, Underscore, Marionette, etc. have become AMD-compatible : they will work as expected without requiring shims. The code is left as is for educational purposes, as you will likely deal with external libraries that will require shims at some point.

How come we require 2 files, but have only one callback argument? Because we only explicitly state the callback arguments we're going to be using (and for which we need a variable to reference). So why bother having the "jquery" as a dependency? Because even though we're not using an explicit callback argument value, we know that within the callback function jQuery has been loaded and can be used via the global $ variable. This is due to jQuery's particularity of registering a global variable we can reference: there's no need to use a callback argument.

Let's move on to loading something with dependencies: Backbone. Here we go:

assets/js/require_main.js

```
1  requirejs.config({
2    baseUrl: "assets/js",
3    paths: {
4      backbone: "vendor/backbone",
5      jquery: "vendor/jquery",
6      json2: "vendor/json2",
7      underscore: "vendor/underscore"
8    },
9
10   shim: {
11     underscore: {
12       exports: "_"
13     },
14     backbone: {
15       deps: ["jquery", "underscore", "json2"],
16       exports: "Backbone"
17     }
18   }
19 });
20
21 require(["backbone"], function(bb){
22   console.log("jQuery version: ", $.fn.jquery);
23   console.log("underscore identity call: ", _.identity(5));
24   console.log("Backbone.history: ", bb.history);
25 });
```

Let's start studying this code in the shim (lines 14-17):

1. we declare a "backbone" key we can then use to require the library;
2. we indicate that Backbone depends on jQuery, Underscore, and JSON2. This implies that before loading Backbone, RequireJS has to load these dependencies;
3. we indicate that the global `Backbone` variable should be returned to the callback function.

Naturally, we need to tell RequireJS where to find the various files so we've had to add paths for Backbone (line 4) and JSON2 (line 6).

With that configuration set up, we can require Backbone on line 21 and use it within the callback function. But how come we can still use jQuery and Underscore without requiring them (lines 22-23)? Because of the dependency tree: Backbone depends on those libraries (see above), so once Backdone

and its dependencies have been loaded we have access to jQuery and Underscore via their global variables (since they're both Backbone dependencies). RequireJS magic in action!

Try adding Marionette to *require_main.js* on your own, then check below. Things to remember:

- You'll need to specify a path for Marionette;
- Marionette depends on Backbone;
- Marionette registers the global `Marionette` variable.

You can see the solution on the next page.

assets/js/require_main.js

```
requirejs.config({
  baseUrl: "assets/js",
  paths: {
    backbone: "vendor/backbone",
    jquery: "vendor/jquery",
    json2: "vendor/json2",
    marionette: "vendor/backbone.marionette",
    underscore: "vendor/underscore"
  },

  shim: {
    underscore: {
      exports: "_"
    },
    backbone: {
      deps: ["jquery", "underscore", "json2"],
      exports: "Backbone"
    },
    marionette: {
      deps: ["backbone"],
      exports: "Marionette"
    }
  }
});

require(["marionette"], function(bbm){
  console.log("jQuery version: ", $.fn.jquery);
  console.log("underscore identity call: ", _.identity(5));
  console.log("Marionette: ", bbm);
});
```

Adding Marionette is pretty straightforward, as it's no different from adding Backbone. However, do note that since Marionette depends on Backbone, we don't need to specify that it also depends (e.g.) on Underscore: RequireJS will load Undercore, then Backbone, then Marionette, due to the dependency relationships we've indicated in the `config` method.

Git commit loading basic libraries:

218e693c31d8e69376a68429e617b4b339fcf58c[16]

[16] https://github.com/davidsulc/structuring-backbone-with-requirejs-and-marionette/commit/218e693c31d8e69376a68429e617b4b339fcf58c