# Make Backbone Better With Extensions

Joe Zimmerman on Mar 18th 2013 with 17 Comments

## Tutorial Details

- 
- **Difficulty:** Intermediate
- **Completion Time:** 30 Minutes

View post on Tuts+ Beta**Tuts+ Beta** is an optimized, mobile-friendly and easy-to-read version of the Tuts+ network.

Backbone is becoming wildly popular as a web application development framework. Along with this popularity comes countless extensions and plugins to enhance the power of the framework, and fill in holes that other felt needed filling. Let's take a look at some of the best choices.

---

## Backbone.Marionette

Developed by Derick Bailey, this extension is quite large and is my personal favorite. Rather than adding one or two features to Backbone, Derick decided to fill in all the biggest holes that he felt existed. Here's what he says about it in the readme file of the GitHub project.

> "Backbone.Marionette is a composite application library for Backbone.js that aims to simplify the construction of large scale JavaScript applications. It is a collection of common design and implementation patterns found in the applications that I (Derick Bailey) have been building with Backbone, and includes various pieces inspired by composite application architectures, such as Microsoft's "Prism" framework."

Let's take a closer look at what Marionette provides us with:

- **Application:** This is a central object that everything in your application can communicate through. It offers a way to set up the main view(s) of your application quickly and easily, a central event hub that every module in your application can communicate through so they aren't dependent on one another, and initializers for fine-grained control of how your application boots up.
- **Modules:** A means of encapsulating module code and namespacing those modules on the central application object.
- **Views:** New view classes to extend that offer native methods for cleaning up, so you don't end up with memory leaks. It also contains rendering boilerplate; for simple views, simply specify the template and model, and it'll handle the rest.
- **Collection/Composite Views:** This is where the "composite application library" bit comes into play. These two views allow you to easily create views that can handle the process of rendering all the views in a collection, or even a nested hierarchy of collections and models, with very little effort.
- **Regions and Layouts:** A region is a object that can handle all the work of rendering, unrendering, and closing views for a particular place in the DOM. A Layout is simply a normal view that also has regions built into it for handling subviews.
- **AppRouter:** A new type of router that can take a controller to handle the

workload so that the router can just contain the configuration of the routes.
- **Events**: Extended from the Wreqr project, Marionette makes Backbone's events even more powerful for creating large-scale event-based applications.

I've only scratched the surface of what Marionette can do. I definitely recommend heading over to GitHub and reading their documentation on the Wiki.

> Additionally, Andrew Burgess covers Marionette in his Tuts+ Premium Advanced Backbone Patterns and Techniques course.

---

## Backbone.Validation

Backbone.Validation is designed to fill a small niche of a problem: namely, model validation. There are several validation extensions for Backbone, but this one seems to have garnered the most respect from the community.

Rather than actually having to write a `validate` method for your models, you can create a `validation` property for your models, which is an object that specifies each of the attributes that you wish to validate and a list of validation rules for each of those attributes. You can also specify error messages for each attribute and pass in custom functions for validating a single attribute. You can see an example below, which is modified from one of the examples on their website.

```
var SomeModel = Backbone.Model.extend({
    validation: {
        name: {
            required: true
        },
        'address.street': {
            required: true
        },
        'address.zip': {
            length: 4
        },
        age: {
            range: [1, 80]
        },
        email: {
            pattern: 'email',
            // supply your own error message
            msg: "Please enter a valid email address"
        },
        // custom validation function for `someAttribute`
        someAttribute: function(value) {
            if(value !== 'somevalue') {
                return 'Error message';
            }
        }
    }
});
```

There are countless built–in validators and patterns that you can check against, and there's even a way to extend the list with your own global validators. This Backbone plugin doesn't quite make validation fun, but it certainly eliminates any excuses for not adding in validation. Please visit their site for more examples and a deeper explanation for how to use this wonderful tool.

---

## Backbone.LayoutManager

Backbone.LayoutManager is all about making Backbone's Views better. Like Backbone.Marionette, it brings in cleanup code to prevent memory leaks, handles all of the boilerplate, and leaves you with just configuration and application–specific code. Unlike Marionette, LayoutManager focuses specifically on Views.

Because LayoutManager focuses solely on Views, it can do more for the views than Marionette does. For instance, LayoutManager is capable of doing asynchronous rendering, in the case that you want to dynamically load your templates from external files.

LayoutManager also can handle subviews, though in a very different way from Marionette. Either way, though, it is largely configuration, which makes things extremely simple and eliminates 90% of the work that you would have needed to do, if you were trying to implement this all on your own. Take a look below for a simple example for adding three subviews to a view:

```
Backbone.Layout.extend({
    views: {
      "header": new HeaderView(),
      "section": new ContentView(),
      "footer": new FooterView()
    }
});
```

As usual, be sure to refer to the GitHub page and documentation to learn more.

---

## Backbone.ModelBinder

Backbone.ModelBinder creates links between the data in your models and markup shown by your views. You can already do this by binding to the change event on your models and rendering the view again, but ModelBinder is more efficient and simpler to use.

Take a look at the code below:

```
var MyView = Backbone.View.extend({
    template: _.template(myTemplate),

    initialize: function() {
        // Old Backbone.js way
        this.model.on('change', this.render, this);
        // or the new Backbone 0.9.10+ way
        this.listenTo(this.model, 'change', this.render);
    },

    render: function() {
        this.$el.html(this.template(this.model.toJSON()));
    }
});
```

The problem with this approach is that any time a single attribute is changed, we need to re-render the entire view. Also, with every render, we need to convert the model to JSON. Finally, if want the binding to be bi-directional (when the model updates, so does the DOM and vice versa), then we need to add in even more logic to the view.

This example was using Underscore's `template` function. Let's assume the template that we passed into it looks like so:

```
<form action="...">
    <label for="firstName">First Name</label>
    <input type="text" id="firstName" name="firstName" value="<%= f

    <label for="lastName">Last Name</label>
    <input type="text" id="lastName" name="lastName" value="<%= las
</form>
```

Using the tags `<%=` and `%>` make the `template` function replace those areas with the `firstName` and `lastName` properties that exist in the JSON that we sent in from the model. We'll assume that the model has both of those attributes, too.

With ModelBinder, instead we can remove those templating tags and send in normal HTML. ModelBinder will see the value of the `name` attribute on the `input` tag, and will automatically assign the model's value for that property to the `value` attribute of the tag. For example, the first `input`'s `name` is set to "firstName". When we use ModelBinder, it'll see that and then set that `input`'s `value` to the model's `firstName` property.

Below, you'll see how our previous example would look after switching to using ModelBinder. Also, realize that the binding is bi-directional, so if the `input` tags are updated, the model will be updated automatically for us.

**HTML Template:**

```
<form action="...">
    <label for="firstName">First Name</label>
    <input type="text" id="firstName" name="firstName">

    <label for="lastName">Last Name</label>
    <input type="text" id="lastName" name="lastName">
</form>
```

JavaScript View:

```
var MyView = Backbone.View.extend({
    template: myTemplate,

    initialize: function() {
        // No more binding in here
    },

    render: function() {
        // Throw the HTML right in
        this.$el.html(this.template);
        // Use ModelBinder to bind the model and view
        modelBinder.bind(this.model, this.el);
    }
});
```

Now we have clean HTML templates, and we only need a single line of code to bind the view's HTML and the models together using `modelBinder.bind`. `modelBinder.bind` takes two required arguments and one optional argument. The first argument is the model with the data that will be bound to the DOM. The second is the DOM node that will be recursively traversed, searching for bindings to make. The final argument is a `binding` object that specifies advanced rules for how bindings should be done, if you don't like the default usage.

ModelBinder can be used on more than just `input` tags. It works on any element. If the element is some type of form input, such as `input`, `select`, or `textarea`, it'll update the values of those element, accordingly. If you bind to an element, such as a `div` or `span`, it will place the model's data between the opening and closing tags of that element (e.g. `<span name="firstName">[data goes here]<span>`).

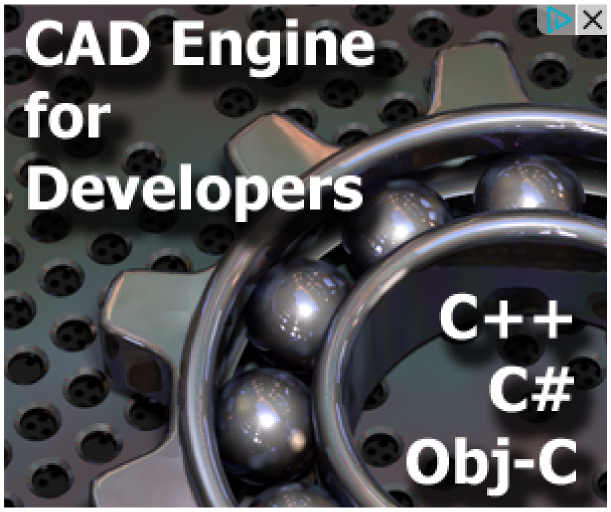
There's plenty more power behind Backbone.ModelBinder than what I've demonstrated here, so be sure to read the documentation on the GitHub repository to learn all about it.

---

## Conclusion

That wraps things up. I’ve only covered a handful of extensions and plugins, but these are what I consider to be of the most use.

The Backbone landscape is constantly changing. If you’d like to view a comprehensive list of the various Backbone extensions that are available, visit this wiki page, which the Backbone team regularly updates.



Tags: backbone.js

**By Joe Zimmerman**
This author has yet to write their bio.

**Note:** Want to add some source code? Type <pre><code> before it and </code></pre> after it. Find out more