Ember.js Guides

Ember.js Guides

The guide for building Ambitious Web Applications

Precious Jahlom Agboado

This book is for sale at http://leanpub.com/emberjsguides

This version was published on 2014-10-04

Contents

Ember.js Guides

Welcome to the Ember.js guides! This documentation will take you from total beginner to Ember expert. It is designed to start from the basics, and slowly increase to more sophisticated concepts until you know everything there is to know about building awesome web applications.

To help you get started, we've also made a 30-minute screencast that will guide you through building a full-featured Ember.js application:

https://www.youtube.com/watch?v=1QHrlFlaXdI[1]

Source code for the app we build in the video is available at https://github.com/tildeio/bloggr-client">https://github.com/tildeio/bloggr-client[2]

Most of these guides are designed to help you start building apps right away. If you'd like to know more about the thinking behind Ember.js, you'll find what you're looking for in the Understanding Ember.js[3] section.

These guides are written in Markdown and are available on GitHub[4], inside the `source/guides` directory. If there is something missing, or you find a typo or mistake, please help us by filing an issue or submitting a pull request. Thanks!

We're excited for all of the great apps you're going to build with Ember.js. To get started, select a topic from the left. They are presented in the order that we think will be most useful to you as you're learning Ember.js, but you can also jump to whatever seems most interesting.

Good luck!

[1]https://www.youtube.com/watch?v=1QHrlFlaXdI

[2]https://github.com/tildeio/bloggr-client">https://github.com/tildeio/bloggr-client

[3]http://emberjs.com/guides/understanding-ember/the-view-layer

[4]https://github.com/emberjs/website/

Getting Started

Welcome to Ember.js! This guide will take you through creating a simple application using Ember.js and briefly explain the core concepts behind the framework. This guide assumes you are already familiar with basic web technologies like JavaScript, HTML, and CSS and development technologies like your browser's web inspector[5].

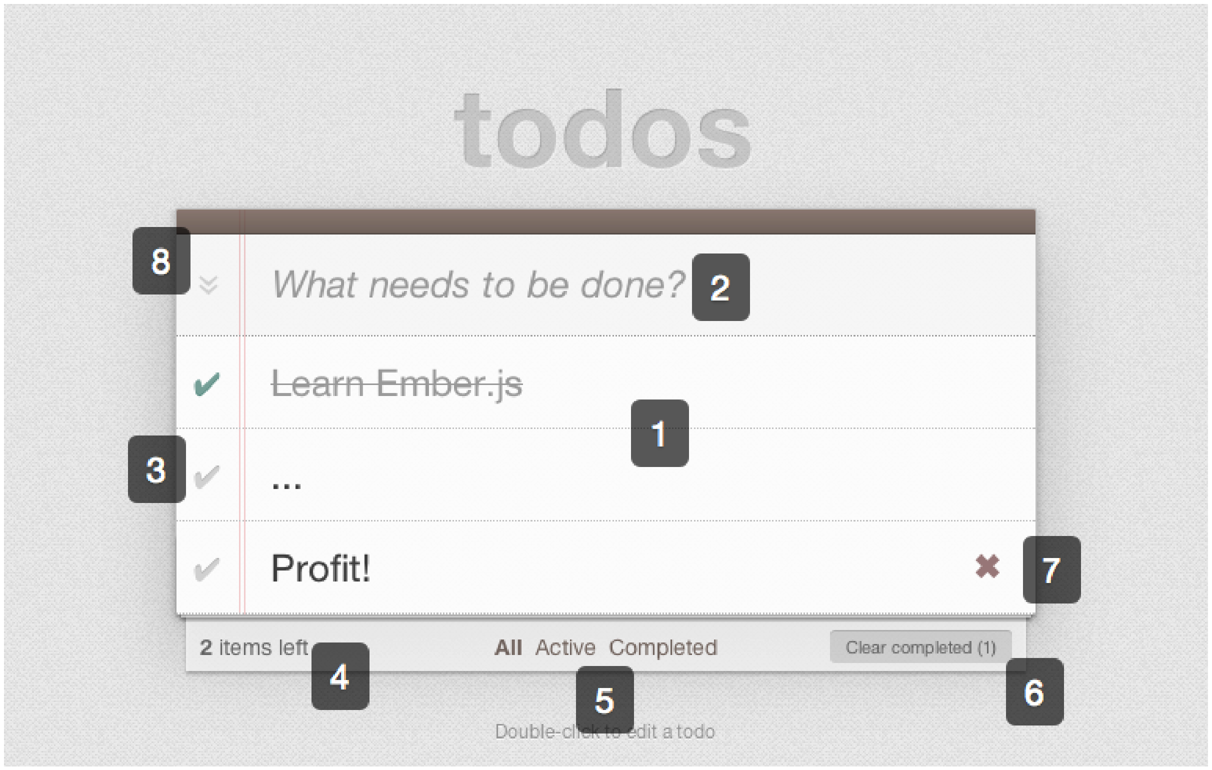
In this guide we will walk through the steps of building the popular TodoMVC demo application[6].

Planning The Application

TodoMVC, despite its small size, contains most of the behaviors typical in modern single page applications. Before continuing, take a moment to understand how TodoMVC works from the user's perspective.

TodoMVC has the following main features:

[5]https://developers.google.com/chrome-developer-tools/
[6]http://todomvc.com



TODO MVC

1. It displays a list of todos for a user to see. This list will grow and shrink as the user adds and removes todos.
2. It accepts text in an `<input>` for entry of new todos. Hitting the `<enter>` key creates the new item and displays it in the list below.
3. It provides a checkbox to toggle between complete and incomplete states for each todo. New todos start as incomplete.
4. It displays the number of incomplete todos and keeps this count updated as new todos are added and existing todos are completed.
5. It provides links for the user to navigate between lists showing all, incomplete, and completed todos.
6. It provides a button to remove all completed todos and informs the user of the number of completed todos. This button will not be visible if there are no completed todos.
7. It provides a button to remove a single specific todo. This button displays as a user hovers over a todo and takes the form of a red X.
8. It provides a checkbox to toggle all existing todos between complete and incomplete states. Further, when all todos are completed this checkbox becomes checked without user interaction.
9. It allows a user to double click to show a textfield for editing a single todo. Hitting the `<enter>` key or moving focus outside of this textfield will persist the changed text.

10. It retains a user's todos between application loads by using the browser's `localstorage` mechanism.

You can interact with a completed version of the application by visiting the TodoMVC site[7].

Creating a Static Mockup

Before adding any code, we can roughly sketch out the layout of our application. In your text editor, create a new file and name it `index.html`. This file will contain the HTML templates of our completed application and trigger requests for the additional image, stylesheet, and JavaScript resources.

To start, add the following text to `index.html`:

```html
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Ember.js • TodoMVC</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <section id="todoapp">
      <header id="header">
        <h1>todos</h1>
        <input type="text" id="new-todo" placeholder="What needs to be done?" />
      </header>

      <section id="main">
        <ul id="todo-list">
          <li class="completed">
            <input type="checkbox" class="toggle">
            <label>Learn Ember.js</label><button class="destroy"></button>
          </li>
          <li>
            <input type="checkbox" class="toggle">
            <label>...</label><button class="destroy"></button>
          </li>
          <li>
            <input type="checkbox" class="toggle">
            <label>Profit!</label><button class="destroy"></button>
```

[7]http://todomvc.com/architecture-examples/emberjs/

```
        </li>
      </ul>

      <input type="checkbox" id="toggle-all">
    </section>

    <footer id="footer">
      <span id="todo-count">
        <strong>2</strong> todos left
      </span>
      <ul id="filters">
        <li>
          <a href="all" class="selected">All</a>
        </li>
        <li>
          <a href="active">Active</a>
        </li>
        <li>
          <a href="completed">Completed</a>
        </li>
      </ul>

      <button id="clear-completed">
        Clear completed (1)
      </button>
    </footer>
  </section>

  <footer id="info">
    <p>Double-click to edit a todo</p>
  </footer>
</body>
</html>
```

The associated stylesheet[8] and background image[9] for this project should be downloaded and placed in the same directory as `index.html`

Open `index.html` in your web browser to ensure that all assets are loading correctly. You should see the TodoMVC application with three hard-coded `<li>` elements where the text of each todo will appear.

[8]http://emberjs.com.s3.amazonaws.com/getting-started/style.css
[9]http://emberjs.com.s3.amazonaws.com/getting-started/bg.png

Live Preview

Ember.js • TodoMVC[10]

Additional Resources

- Changes in this step in `diff` format[11]
- TodoMVC stylesheet[12]
- TodoMVC background image[13]

Obtaining Ember.Js And Dependencies

TodoMVC has a few dependencies:

- jQuery[14]
- Handlebars[15]
- Ember.js 1.3[16]
- Ember Data 1.0 beta[17]

For this example, all of these resources should be stored in the folder `js/libs` located in the same location as `index.html`. Update your `index.html` to load these files by placing `<script>` tags just before your closing `</body>` tag in the following order:

```
<!-- ... additional lines truncated for brevity ... -->
  <script src="js/libs/jquery-1.10.2.min.js"></script>
  <script src="js/libs/handlebars-1.0.0.js"></script>
  <script src="js/libs/ember.js"></script>
  <script src="js/libs/ember-data.js"></script>
</body>
<!-- ... additional lines truncated for brevity ... -->
```

Reload your web browser to ensure that all files have been referenced correctly and no errors occur.

If you are using a package manager, such as bower[18], make sure to checkout the Getting Ember[19] guide for info on other ways to get Ember.js (this guide is dependant on ember-data v1.0 or greater so please be sure to use the latest beta).

[10]http://jsbin.com/uduyip
[11]https://github.com/emberjs/quickstart-code-sample/commit/4d91f9fa1f6be4f4675b54babd3074550095c930
[12]http://emberjs.com.s3.amazonaws.com/getting-started/style.css
[13]http://emberjs.com.s3.amazonaws.com/getting-started/bg.png
[14]http://code.jquery.com/jquery-1.10.2.min.js
[15]http://builds.handlebarsjs.com.s3.amazonaws.com/handlebars-1.0.0.js
[16]http://builds.emberjs.com/tags/v1.3.0/ember.js
[17]http://builds.emberjs.com/tags/v1.0.0-beta.5/ember-data.js
[18]http://bower.io
[19]http://emberjs.com/guides/getting-ember

Live Preview

Ember.js • TodoMVC[20]

Additional Resources

- Changes in this step in `diff` format[21]

Next, we will create an Ember.js application, a route ('/'), and convert our static mockup into a Handlebars template.

Inside your `js` directory, add a file for the application at `js/application.js` and a file for the router at `js/router.js`. You may place these files anywhere you like (even just putting all code into the same file), but this guide will assume you have separated them into their own files and named them as indicated.

Inside `js/application.js` add the following code:

```
window.Todos = Ember.Application.create();
```

This will create a new instance of `Ember.Application` and make it available as a variable named `Todos` within your browser's JavaScript environment.

Inside `js/router.js` add the following code:

```
Todos.Router.map(function() {
  this.resource('todos', { path: '/' });
});
```

This will tell Ember.js to detect when the application's URL matches `'/'` and to render the `todos` template.

Next, update your `index.html` to wrap the inner contents of `<body>` in a Handlebars script tag and include `js/application.js` and `js/router.js` after Ember.js and other javascript dependencies:

[20] http://jsbin.com/ijefig
[21] https://github.com/emberjs/quickstart-code-sample/commit/0880d6e21b83d916a02fd17163f58686a37b5b2c

```
<!-- ... additional lines truncated for brevity ... -->
<body>
  <script type="text/x-handlebars" data-template-name="todos">

    <section id="todoapp">
      {{! ... additional lines truncated for brevity ... }}
    </section>

    <footer id="info">
      <p>Double-click to edit a todo</p>
    </footer>

  </script>

  <!-- ... Ember.js and other javascript dependencies ... -->
  <script src="js/application.js"></script>
  <script src="js/router.js"></script>
</body>
<!-- ... additional lines truncated for brevity ... -->
```

Reload your web browser to ensure that all files have been referenced correctly and no errors occur.

Live Preview

Ember.js • TodoMVC[22]

Additional Resources

- Changes in this step in `diff` format[23]
- Handlebars Guide[24]
- Ember.Application Guide[25]
- Ember.Application API Documentation[26]

Modeling Data

Next we will create a model class to describe todo items.

Create a file at `js/models/todo.js` and put the following code inside:

[22] http://jsbin.com/OKEMIJi
[23] https://github.com/emberjs/quickstart-code-sample/commit/8775d1bf4c05eb82adf178be4429e5b868ac145b
[24] http://emberjs.com/guides/templates/handlebars-basics
[25] http://emberjs.com/guides/application
[26] http://emberjs.com/api/classes/Ember.Application.html

```
Todos.Todo = DS.Model.extend({
  title: DS.attr('string'),
  isCompleted: DS.attr('boolean')
});
```

This code creates a new class `Todo` and places it within your application's namespace. Each todo will have two attributes: `title` and `isCompleted`.

You may place this file anywhere you like (even just putting all code into the same file), but this guide will assume you have created a file and named it as indicated.

Finally, update your `index.html` to include a reference to this new file:

```
<!-- ... additional lines truncated for brevity ... -->
  <script src="js/models/todo.js"></script>
</body>
<!-- ... additional lines truncated for brevity ... -->
```

Reload your web browser to ensure that all files have been referenced correctly and no errors occur.

Live Preview

Ember.js • TodoMVC[27]

Additional Resources

- Changes in this step in `diff` format[28]
- Models Guide[29]

Using Fixtures

Now we'll add fixture data. Fixtures are a way to put sample data into an application before connecting the application to long-term persistence.

First, update `js/application.js` to indicate that your application's `ApplicationAdapter` is an extension of the `DS.FixtureAdapter`. Adapters are responsible for communicating with a source of data for your application. Typically this will be a web service API, but in this case we are using an adapter designed to load fixture data:

[27]http://jsbin.com/AJoyOGo
[28]https://github.com/emberjs/quickstart-code-sample/commit/a1ccdb43df29d316a7729321764c00b8d850fcd1
[29]http://emberjs.com/guides/models

```
window.Todos = Ember.Application.create();

Todos.ApplicationAdapter = DS.FixtureAdapter.extend();
```

Next, update the file at `js/models/todo.js` to include the following fixture data:

```
// ... additional lines truncated for brevity ...
Todos.Todo.FIXTURES = [
 {
   id: 1,
   title: 'Learn Ember.js',
   isCompleted: true
 },
 {
   id: 2,
   title: '...',
   isCompleted: false
 },
 {
   id: 3,
   title: 'Profit!',
   isCompleted: false
 }
];
```

Reload your web browser to ensure that all files have been referenced correctly and no errors occur.

Live Preview

Ember.js • TodoMVC[30]

Additional Resources

- Changes in this step in `diff` format[31]

Displaying Model Data

Next we'll update our application to display dynamic todos, replacing our hard coded section in the `todos` template.

Inside the file `js/router.js` implement a `TodosRoute` class with a `model` function that returns all the existing todos:

[30]http://jsbin.com/Ovuw
[31]https://github.com/emberjs/quickstart-code-sample/commit/a586fc9de92cad626ea816e9bb29445525678098

```
// ... additional lines truncated for brevity ...
Todos.TodosRoute = Ember.Route.extend({
  model: function() {
    return this.store.find('todo');
  }
});
```

Because we hadn't implemented this class before, Ember.js provided a `Route` for us with the default behavior of rendering a matching template named `todos` using its naming conventions for object creation[32].

Now that we need custom behavior (returning a specific set of models), we implement the class and add the desired behavior.

Update `index.html` to replace the static `<li>` elements with a Handlebars `{{each}}` helper and a dynamic `{{title}}` for each item.

```
{{! ... additional lines truncated for brevity ... }}
<ul id="todo-list">
  {{#each}}
    <li>
      <input type="checkbox" class="toggle">
      <label>{{title}}</label><button class="destroy"></button>
    </li>
  {{/each}}
</ul>
{{! ... additional lines truncated for brevity ... }}
```

The template loops over the content of its controller. This controller is an instance of `ArrayController` that Ember.js has provided for us as the container for our models. Because we don't need custom behavior for this object yet, we can use the default object provided by the framework.

Reload your web browser to ensure that all files have been referenced correctly and no errors occur.

Live Preview

Ember.js • TodoMVC[33]

[32] http://emberjs.com/guides/concepts/naming-conventions/
[33] http://jsbin.com/EJISAne

Additional Resources

- Changes in this step in `diff` format[34]
- Templates Guide[35]
- Controllers Guide[36]
- Naming Conventions Guide[37]

Displaying A Model's Complete State

TodoMVC strikes through completed todos by applying a CSS class `completed` to the `<li>` element. Update `index.html` to apply a CSS class to this element when a todo's `isCompleted` property is true:

```
{{! ... additional lines truncated for brevity ... }}
<li {{bind-attr class="isCompleted:completed"}}>
  <input type="checkbox" class="toggle">
  <label>{{title}}</label><button class="destroy"></button>
</li>
{{! ... additional lines truncated for brevity ... }}
```

This code will apply the CSS class `completed` when the todo's `isCompleted` property is `true` and remove it when the property becomes `false`.

The first fixture todo in our application has an `isCompleted` property of `true`. Reload the application to see the first todo is now decorated with a strike-through to visually indicate it has been completed.

Live Preview

Ember.js • TodoMVC[38]

Additional Resources

- Changes in this step in `diff` format[39]
- bind-attr API documentation[40]
- bind and bind-attr article by Peter Wagenet[41]

[34] https://github.com/emberjs/quickstart-code-sample/commit/87bd57700110d9dd0b351c4d4855edf90baac3a8
[35] http://emberjs.com/guides/templates/handlebars-basics
[36] http://emberjs.com/guides/controllers
[37] http://emberjs.com/guides/concepts/naming-conventions
[38] http://jsbin.com/oKuwomo
[39] https://github.com/emberjs/quickstart-code-sample/commit/b15e5deffc41cf5ba4161808c7f46a283dc2277f
[40] http://emberjs.com/api/classes/Ember.Handlebars.helpers.html#method_bind-attr
[41] http://www.emberist.com/2012/04/06/bind-and-bindattr.html

Creating A New Model Instance

Next we'll update our static HTML `<input>` to an Ember view that can expose more complex behaviors. Update `index.html` to replace the new todo `<input>` with an `{{input}}` helper:

```
{{! ... additional lines truncated for brevity ... }}
<h1>todos</h1>
{{input type="text" id="new-todo" placeholder="What needs to be done?"
              value=newTitle action="createTodo"}}
{{! ... additional lines truncated for brevity ... }}
```

This will render an `<input>` element at this location with the same `id` and `placeholder` attributes applied. It will also connect the `newTitle` property of this template's controller to the `value` attribute of the `<input>`. When one changes, the other will automatically update to remain synchronized.

Additionally, we connect user interaction (pressing the `<enter>` key) to a method `createTodo` on this template's controller.

Because we have not needed a custom controller behavior until this point, Ember.js provided a default controller object for this template. To handle our new behavior, we can implement the controller class Ember.js expects to find according to its naming conventions[42] and add our custom behavior. This new controller class will automatically be associated with this template for us.

Add a `js/controllers/todos_controller.js` file. You may place this file anywhere you like (even just putting all code into the same file), but this guide will assume you have created the file and named it as indicated.

Inside `js/controllers/todos_controller.js` implement the controller Ember.js expects to find according to its naming conventions[43]:

```
Todos.TodosController = Ember.ArrayController.extend({
  actions: {
    createTodo: function() {
      // Get the todo title set by the "New Todo" text field
      var title = this.get('newTitle');
      if (!title) { return false; }
      if (!title.trim()) { return; }

      // Create the new Todo model
      var todo = this.store.createRecord('todo', {
        title: title,
        isCompleted: false
```

[42]http://emberjs.com/guides/concepts/naming-conventions

[43]http://emberjs.com/guides/concepts/naming-conventions

```
      });

      // Clear the "New Todo" text field
      this.set('newTitle', '');

      // Save the new model
      todo.save();
    }
  }
});
```

This controller will now respond to user action by using its `newTitle` property as the title of a new todo whose `isCompleted` property is false. Then it will clear its `newTitle` property which will synchronize to the template and reset the textfield. Finally, it persists any unsaved changes on the todo.

In `index.html` include `js/controllers/todos_controller.js` as a dependency:

```
<!--- ... additional lines truncated for brevity ... -->
   <script src="js/models/todo.js"></script>
   <script src="js/controllers/todos_controller.js"></script>
 </body>
 <!--- ... additional lines truncated for brevity ... -->
```

Reload your web browser to ensure that all files have been referenced correctly and no errors occur. You should now be able to add additional todos by entering a title in the `<input>` and hitting the `<enter>` key.

Live Preview

Ember.js • TodoMVC[44]

Additional Resources

- Changes in this step in `diff` format[45]
- Ember.TextField API documention[46]
- Ember Controller Guide[47]
- Naming Conventions Guide[48]

[44]href="http://jsbin.com/ImukUZO
[45]https://github.com/emberjs/quickstart-code-sample/commit/60feb5f369c8eecd9df3f561fbd01595353ce803
[46]http://emberjs.com/api/classes/Ember.TextField.html
[47]http://emberjs.com/guides/controllers
[48]http://emberjs.com/guides/concepts/naming-conventions

Marking a Model as Complete or Incomplete

In this step we'll update our application to allow a user to mark a todo as complete or incomplete and persist the updated information.

In `index.html` update your template to wrap each todo in its own controller by adding an `itemController` argument to the `{{each}}` Handlebars helper. Then convert our static `<input type="checkbox">` into a `{{input}}` helper:

```
{{! ... additional lines truncated for brevity ... }}
{{#each itemController="todo"}}
  <li {{bind-attr class="isCompleted:completed"}}>
    {{input type="checkbox" checked=isCompleted class="toggle"}}
    <label>{{title}}</label><button class="destroy"></button>
  </li>
{{/each}}
{{! ... additional lines truncated for brevity ... }}
```

When this `{{input}}` is rendered it will ask for the current value of the controller's `isCompleted` property. When a user clicks this input, it will set the value of the controller's `isCompleted` property to either `true` or `false` depending on the new checked value of the input.

Implement the controller for each todo by matching the name used as the `itemController` value to a class in your application `Todos.TodoController`. Create a new file at `js/controllers/todo_-controller.js` for this code. You may place this file anywhere you like (even just putting all code into the same file), but this guide will assume you have created the file and named it as indicated.

Inside `js/controllers/todo_controller.js` add code for `Todos.TodoController` and its `isCompleted` property:

```
Todos.TodoController = Ember.ObjectController.extend({
  isCompleted: function(key, value){
    var model = this.get('model');

    if (value === undefined) {
      // property being used as a getter
      return model.get('isCompleted');
    } else {
      // property being used as a setter
      model.set('isCompleted', value);
      model.save();
      return value;
    }
  }.property('model.isCompleted')
});
```

When called from the template to display the current `isCompleted` state of the todo, this property will proxy that question[49] to its underlying `model`. When called with a value because a user has toggled the checkbox in the template, this property will set the `isCompleted` property of its `model` to the passed value (`true` or `false`), persist the model update, and return the passed value so the checkbox will display correctly.

The `isCompleted` function is marked a computed property[50] whose value is dependent on the value of `model.isCompleted`.

In `index.html` include `js/controllers/todo_controller.js` as a dependency:

```
<!--- ... additional lines truncated for brevity ... -->
   <script src="js/models/todo.js"></script>
   <script src="js/controllers/todos_controller.js"></script>
   <script src="js/controllers/todo_controller.js"></script>
 </body>
 <!--- ... additional lines truncated for brevity ... -->
```

Reload your web browser to ensure that all files have been referenced correctly and no errors occur. You should now be able to change the `isCompleted` property of a todo.

Live Preview

Ember.js • TodoMVC[51]

Additional Resources

- Changes in this step in `diff` format[52]
- Ember.Checkbox API documentation[53]
- Ember Controller Guide[54]
- Computed Properties Guide[55]
- Naming Conventions Guide[56]

[49]http://emberjs.com/api/classes/Ember.ObjectController.html
[50]http://emberjs.com/guides/object-model/computed-properties/
[51]http://jsbin.com/UDoPajA
[52]https://github.com/emberjs/quickstart-code-sample/commit/8d469c04c237f39a58903a3856409a2592cc18a9
[53]/api/classes/Ember.Checkbox.html
[54]http://emberjs.com/guides/controllers
[55]http://emberjs.com/guides/object-model/computed-properties/
[56]http://emberjs.com/guides/concepts/naming-conventions

Displaying the Number of Incomplete Todos

Next we'll update our template's hard-coded count of completed todos to reflect the actual number of completed todos. Update `index.html` to use two properties:

```
{{! ... additional lines truncated for brevity ... }}
<span id="todo-count">
  <strong>{{remaining}}</strong> {{inflection}} left
</span>
{{! ... additional lines truncated for brevity ... }}
```

Implement these properties as part of this template's controller, the `Todos.TodosController`:

```
// ... additional lines truncated for brevity ...
actions: {
  // ... additional lines truncated for brevity ...
},

remaining: function() {
  return this.filterBy('isCompleted', false).get('length');
}.property('@each.isCompleted'),

inflection: function() {
  var remaining = this.get('remaining');
  return remaining === 1 ? 'item' : 'items';
}.property('remaining')
// ... additional lines truncated for brevity ...
```

The `remaining` property will return the number of todos whose `isCompleted` property is false. If the `isCompleted` value of any todo changes, this property will be recomputed. If the value has changed, the section of the template displaying the count will be automatically updated to reflect the new value.

The `inflection` property will return either a plural or singular version of the word "item" depending on how many todos are currently in the list. The section of the template displaying the count will be automatically updated to reflect the new value.

Reload your web browser to ensure that no errors occur. You should now see an accurate number for remaining todos.

Live Preview

Ember.js • TodoMVC[57]

[57] http://jsbin.com/onOCIrA

Additional Resources

- Changes in this step in `diff` format[58]
- Computed Properties Guide[59]

Toggling Between Showing and Editing States

TodoMVC allows users to double click each todo to display a text `<input>` element where the todo's title can be updated. Additionally the `<li>` element for each todo obtains the CSS class `editing` for style and positioning.

We'll update the application to allow users to toggle into this editing state for a todo. In `index.html` update the contents of the `{{each}}` Handlebars helper to:

```
 {{! ... additional lines truncated for brevity ... }}
{{#each itemController="todo"}}
  <li {{bind-attr class="isCompleted:completed isEditing:editing"}}>
    {{#if isEditing}}
      <input class="edit">
    {{else}}
      {{input type="checkbox" checked=isCompleted class="toggle"}}
      <label {{action "editTodo" on="doubleClick"}}>{{title}}</label><button cla\
ss="destroy"></button>
    {{/if}}
  </li>
{{/each}}
 {{! ... additional lines truncated for brevity ... }}
```

The above code applies three new behaviors to our application: it applies the CSS class `editing` when the controller's `isEditing` property is true and removes it when the `isEditing` property is false. We add a new `{{action}}` helper to the `<label>` so double-clicks will call `editTodo` on this todo's controller. Finally, we wrap our todo in a Handlebars `{{if}}` helper so a text `<input>` will display when we are editing and the todos title will display when we are not editing.

Inside `js/controllers/todo_controller.js` we'll implement the matching logic for this template behavior:

[58] https://github.com/emberjs/quickstart-code-sample/commit/b418407ed9666714c82d894d6b70f785674f7a45
[59] http://emberjs.com/guides/object-model/computed-properties/

```
Todos.TodoController = Ember.ObjectController.extend({
  actions: {
    editTodo: function() {
      this.set('isEditing', true);
    }
  },

  isEditing: false,

// ... additional lines truncated for brevity ...
```

Above we defined an initial `isEditing` value of `false` for controllers of this type and said that when the `editTodo` action is called it should set the `isEditing` property of this controller to `true`. This will automatically trigger the sections of template that use `isEditing` to update their rendered content.

Reload your web browser to ensure that no errors occur. You can now double-click a todo to edit it.

Live Preview

Ember.js • TodoMVC[60]

Additional Resources

- Changes in this step in `diff` format[61]
- Handlebars Conditionals Guide[62]
- bind-attr API documentation[63]
- action API documentation[64]
- bind and bindAttr article by Peter Wagenet[65]

Accepting Edits

In the previous step we updated TodoMVC to allow a user to toggle the display of a text `<input>` for editing a todo's title. Next, we'll add the behavior that immediately focuses the `<input>` when it appears, accepts user input and, when the user presses the `<enter>` key or moves focus away from the editing `<input>` element, persists these changes, then redisplays the todo with its newly updated text.

[60]http://jsbin.com/usiXemu
[61]https://github.com/emberjs/quickstart-code-sample/commit/616bc4f22900bbaa2bf9bdb8de53ba41209d8cc0
[62]http://emberjs.com/guides/templates/conditionals
[63]http://emberjs.com/api/classes/Ember.Handlebars.helpers.html#method_bind-attr
[64]http://emberjs.com/api/classes/Ember.Handlebars.helpers.html#method_action
[65]http://www.emberist.com/2012/04/06/bind-and-bindattr.html

To accomplish this, we'll create a new custom component and register it with Handlebars to make it available to our templates.

Create a new file `js/views/edit_todo_view.js`. You may place this file anywhere you like (even just putting all code into the same file), but this guide will assume you have created the file and named it as indicated.

In `js/views/edit_todo_view.js` create an extension of `Ember.TextField` and register it as a helper[66]:

```
Todos.EditTodoView = Ember.TextField.extend({
  didInsertElement: function() {
    this.$().focus();
  }
});

Ember.Handlebars.helper('edit-todo', Todos.EditTodoView);
```

In `index.html` require this new file:

```
<!--- ... additional lines truncated for brevity ... -->
  <script src="js/controllers/todo_controller.js"></script>
  <script src="js/views/edit_todo_view.js"></script>
</body>
<!--- ... additional lines truncated for brevity ... -->
```

In `index.html` replace the static `<input>` element with our custom `{{edit-todo}}` component, connecting the `value` property, and actions:

```
{{! ... additional lines truncated for brevity ... }}
{{#if isEditing}}
  {{edit-todo class="edit" value=title focus-out="acceptChanges"
                           insert-newline="acceptChanges"}}
{{else}}
{{! ... additional lines truncated for brevity ... }}
```

Pressing the `<enter>` key will trigger the `acceptChanges` event on the instance of `TodoController`. Moving focus away from the `<input>` will trigger the `focus-out` event, calling a method `acceptChanges` on this view's instance of `TodoController`.

[66]http://emberjs.com/api/classes/Ember.Handlebars.html#method_helper

Additionally, we connect the `value` property of this `<input>` to the `title` property of this instance of `TodoController`. We will not implement a `title` property on the controller so it will retain the default behavior of proxying all requests[67] to its `model`.

A CSS class `edit` is applied for styling.

In `js/controllers/todo_controller.js`, add the method `acceptChanges` that we called from `EditTodoView`:

```
// ... additional lines truncated for brevity ...
actions: {
  editTodo: function() {
    this.set('isEditing', true);
  },
  acceptChanges: function() {
    this.set('isEditing', false);

    if (Ember.isEmpty(this.get('model.title'))) {
      this.send('removeTodo');
    } else {
      this.get('model').save();
    }
  },
  removeTodo: function () {
    var todo = this.get('model');
    todo.deleteRecord();
    todo.save();
  }
},
// ... additional lines truncated for brevity ...
```

This method will set the controller's `isEditing` property to false and commit all changes made to the todo.

Live Preview

Ember.js • TodoMVC[68]

Additional Resources

- Changes in this step in `diff` format[69]

[67]http://emberjs.com/guides/controllers/#toc_representing-models
[68]http://jsbin.com/USOlAna
[69]https://github.com/emberjs/quickstart-code-sample/commit/a7e2f40da4d75342358acdfcbda7a05ccc90f348

- Controller Guide[70]
- Ember.TextField API documentation[71]

Deleting a Model

TodoMVC displays a button for removing todos next to each todo when its `<li>` is hovered. Clicking this button will remove the todo and update the display of remaining incomplete todos and remaining completed todos appropriately.

In `index.html` update the static `<button>` element to include an `{{action}}` Handlebars helper:

```
{{! ... additional lines truncated for brevity ... }}
<button {{action "removeTodo"}} class="destroy"></button>
{{! ... additional lines truncated for brevity ... }}
```

This will call the `removeTodo` action defined in the previous chapter and will delete the todo locally and then persist this data change.

Because the todo is no longer part of the collection of all todos, its `<li>` element in the page will be automatically removed for us. If the deleted todo was incomplete, the count of remaining todos will be decreased by one and the display of this number will be automatically re-rendered. If the new count results in an inflection change between “todo” and “todos” this area of the page will be automatically re-rendered.

Reload your web browser to ensure that there are no errors and the behaviors described above occurs.

Live Preview

Ember.js • TodoMVC[72]

Additional Resources

- Changes in this step in `diff` format[73]
- action API documentation[74]

[70] http://emberjs.com/guides/controllers
[71] http://emberjs.com/api/classes/Ember.TextField.html
[72] http://jsbin.com/eREkanA
[73] https://github.com/emberjs/quickstart-code-sample/commit/14e1f129f76bae8f8ea6a73de1e24d810678a8fe
[74] http://emberjs.com/api/classes/Ember.Handlebars.helpers.html#method_action

Adding Child Routes

Next we will split our single template into a set of nested templates so we can transition between different lists of todos in reaction to user interaction.

In `index.html` move the entire `<ul>` of todos into a new template named `todos/index` by adding a new Handlebars template `<script>` tag inside the `<body>` of the document:

```
<!--- ... additional lines truncated for brevity ... -->
<body>
<script type="text/x-handlebars" data-template-name="todos/index">
  <ul id="todo-list">
    {{#each itemController="todo"}}
      <li {{bind-attr class="isCompleted:completed isEditing:editing"}}>
        {{#if isEditing}}
          {{edit-todo class="edit" value=title focus-out="acceptChanges" insert-\
newline="acceptChanges"}}
        {{else}}
          {{input type="checkbox" checked=isCompleted class="toggle"}}
          <label {{action "editTodo" on="doubleClick"}}>{{title}}</label><button\
 {{action "removeTodo"}} class="destroy"></button>
        {{/if}}
      </li>
    {{/each}}
  </ul>
</script>
<!--- ... additional lines truncated for brevity ... -->
```

Still within `index.html` place a Handlebars `{{outlet}}` helper where the `<ul>` was previously:

```
{{! ... additional lines truncated for brevity ... }}
<section id="main">
  {{outlet}}

  <input type="checkbox" id="toggle-all">
</section>
{{! ... additional lines truncated for brevity ... }}
```

The `{{outlet}}` Handlebars helper designates an area of a template that will dynamically update as we transition between routes. Our first new child route will fill this area with the list of all todos in the application.

In `js/router.js` update the router to change the `todos` mapping, with an additional empty function parameter so it can accept child routes, and add this first `index` route:

```
Todos.Router.map(function () {
  this.resource('todos', { path: '/' }, function () {
    // additional child routes will go here later
  });
});

// ... additional lines truncated for brevity ...

Todos.TodosIndexRoute = Ember.Route.extend({
  model: function() {
    return this.modelFor('todos');
  }
});
```

When the application loads at the url `'/'` Ember.js will enter the `todos` route and render the `todos` template as before. It will also transition into the `todos.index` route and fill the `{{outlet}}` in the `todos` template with the `todos/index` template. The model data for this template is the result of the `model` method of `TodosIndexRoute`, which indicates that the model for this route is the same model as for the `TodosRoute`.

This mapping is described in more detail in the Naming Conventions Guide[75].

Live Preview

Ember.js • TodoMVC[76]

Additional Resources

- Changes in this step in `diff` format[77]
- Ember Router Guide[78]
- Ember Controller Guide[79]
- outlet API documentation[80]

Transitioning to Show Only Incomplete Todos

Next we'll update the application so a user can navigate to a url where only todos that are not complete are displayed.

[75] http://emberjs.com/guides/concepts/naming-conventions
[76] http://jsbin.com/oweNovo
[77] https://github.com/emberjs/quickstart-code-sample/commit/3bab8f1519ffc1ca2d5a12d1de35e4c764c91f05
[78] http://emberjs.com/guides/routing
[79] http://emberjs.com/guides/controllers
[80] http://emberjs.com/api/classes/Ember.Handlebars.helpers.html#method_outlet

In `index.html` convert the `<a>` tag for ‘Active’ todos into a Handlebars `{{link-to}}` helper and remove the active class from the `<a>` tag for ‘All’:

```
{{! ... additional lines truncated for brevity ... }}
<li>
  <a href="all">All</a>
</li>
<li>
  {{#link-to "todos.active" activeClass="selected"}}Active{{/link-to}}
</li>
<li>
  <a href="completed">Completed</a>
</li>
{{! ... additional lines truncated for brevity ... }}
```

In `js/router.js` update the router to recognize this new path and implement a matching route:

```
Todos.Router.map(function() {
  this.resource('todos', { path: '/' }, function() {
    // additional child routes
    this.route('active');
  });
});

// ... additional lines truncated for brevity ...
Todos.TodosActiveRoute = Ember.Route.extend({
  model: function(){
    return this.store.filter('todo', function(todo) {
      return !todo.get('isCompleted');
    });
  },
  renderTemplate: function(controller) {
    this.render('todos/index', {controller: controller});
  }
});
```

The model data for this route is the collection of todos whose `isCompleted` property is `false`. When a todo’s `isCompleted` property changes this collection will automatically update to add or remove the todo appropriately.

Normally transitioning into a new route changes the template rendered into the parent `{{outlet}}`, but in this case we’d like to reuse the existing `todos/index` template. We can accomplish this by implementing the `renderTemplate` method and calling `render` ourselves with the specific template and controller options.

Reload your web browser to ensure that there are no errors and the behavior described above occurs.

Live Preview

[Ember.js • TodoMVC[(http://jsbin.com/arITiZu)

Additional Resources

- Changes in this step in `diff` format[81]
- link-to API documentation[82]
- Route#renderTemplate API documentation[83]
- Route#render API documentation[84]
- Ember Router Guide[85]

Transitioning to Show Only Complete Todos

Next we'll update the application so a user can navigate to a url where only todos that have already been completed are displayed.

In `index.html` convert the `<a>` tag for 'Completed' todos into a Handlebars `{{link-to}}` helper:

```
{{! ... additional lines truncated for brevity ... }}
<li>
  <a href="all">All</a>
</li>
<li>
  {{#link-to "todos.active" activeClass="selected"}}Active{{/link-to}}
</li>
<li>
  {{#link-to "todos.completed" activeClass="selected"}}Completed{{/link-to}}
</li>
{{! ... additional lines truncated for brevity ... }}
```

In `js/router.js` update the router to recognize this new path and implement a matching route:

[81]https://github.com/emberjs/quickstart-code-sample/commit/2a1d35293a52e40d0125f552a1a8b2c01f759313
[82]http://emberjs.com/api/classes/Ember.Handlebars.helpers.html#method_link-to
[83]http://emberjs.com/api/classes/Ember.Route.html#method_renderTemplate
[84]http://emberjs.com/api/classes/Ember.Route.html#method_render
[85]http://emberjs.com/guides/routing

```
Todos.Router.map(function() {
  this.resource('todos', { path: '/' }, function() {
    // additional child routes
    this.route('active');
    this.route('completed');
  });
});

// ... additional lines truncated for brevity ...

Todos.TodosCompletedRoute = Ember.Route.extend({
  model: function() {
    return this.store.filter('todo', function(todo) {
      return todo.get('isCompleted');
    });
  },
  renderTemplate: function(controller) {
    this.render('todos/index', {controller: controller});
  }
});
```

The model data for this route is the collection of todos whose `isCompleted` property is `true`. Just like we recently saw with the similar function for the active todos, changes to a todo's `isCompleted` property will automatically cause this collection to refresh, updating the UI accordingly.

`TodosCompletedRoute` has a similar purpose to the active todos - to reuse the existing `todos/index` template, rather than having to create a new template.

Reload your web browser to ensure that there are no errors and the behavior described above occurs.

Live Preview

Ember.js • TodoMVC[86]

Additional Resources

- Changes in this step in `diff` format[87]
- link-to API documentation[88]
- Route#renderTemplate API documentation[89]

[86]http://jsbin.com/OzUvuPu
[87]https://github.com/emberjs/quickstart-code-sample/commit/bba939a11197552e3a927bcb3a3adb9430e4f331
[88]http://emberjs.com/api/classes/Ember.Handlebars.helpers.html#method_link-to
[89]http://emberjs.com/api/classes/Ember.Route.html#method_renderTemplate

- Route#render API documentation[90]
- Ember Router Guide[91]

Transitioning back to Show All Todos

Next we can update the application to allow navigating back to the list of all todos.

In `index.html` convert the `<a>` tag for 'All' todos into a Handlebars `{{link-to}}` helper:

```
{{! ... additional lines truncated for brevity ... }}
<li>
  {{#link-to "todos.index" activeClass="selected"}}All{{/link-to}}
</li>
<li>
  {{#link-to "todos.active" activeClass="selected"}}Active{{/link-to}}
</li>
<li>
  {{#link-to "todos.completed" activeClass="selected"}}Completed{{/link-to}}
</li>
{{! ... additional lines truncated for brevity ... }}
```

Reload your web browser to ensure that there are no errors. You should be able to navigate between urls for all, active, and completed todos.

Live Preview

Ember.js • TodoMVC[92]

Additional Resources

- Changes in this step in `diff` format[93]
- link-to API documentation[94]

[90]http://emberjs.com/api/classes/Ember.Route.html#method_render
[91]http://emberjs.com/guides/routing
[92]http://jsbin.com/uYuGA
[93]https://github.com/emberjs/quickstart-code-sample/commit/843ff914873081560e4ba97df0237b8595b6ae51
[94]http://emberjs.com/api/classes/Ember.Handlebars.helpers.html#method_link-to

Displaying a Button to Remove All Completed Todos

TodoMVC allows users to delete all completed todos at once by clicking a button. This button is visible only when there are any completed todos, displays the number of completed todos, and removes all completed todos from the application when clicked.

In this step, we'll implement that behavior. In `index.html` update the static `<button>` for clearing all completed todos:

```
{{! ... additional lines truncated for brevity ... }}
{{#if hasCompleted}}
  <button id="clear-completed" {{action "clearCompleted"}}>
    Clear completed ({{completed}})
  </button>
{{/if}}
{{! ... additional lines truncated for brevity ... }}
```

In `js/controllers/todos_controller.js` implement the matching properties and a method that will clear completed todos and persist these changes when the button is clicked:

```
// ... additional lines truncated for brevity ...
actions: {
  clearCompleted: function() {
    var completed = this.filterBy('isCompleted', true);
    completed.invoke('deleteRecord');
    completed.invoke('save');
  },
  // ... additional lines truncated for brevity ...
},
hasCompleted: function() {
  return this.get('completed') > 0;
}.property('completed'),

completed: function() {
  return this.filterBy('isCompleted', true).get('length');
}.property('@each.isCompleted'),
// ... additional lines truncated for brevity ...
```

The `completed` and `clearCompleted` methods both invoke the `filterBy` method, which is part of the ArrayController[95] API and returns an instance of EmberArray[96] which contains only the items

[95]http://emberjs.com/api/classes/Ember.ArrayController.html#method_filterProperty
[96]http://emberjs.com/api/classes/Ember.Array.html

for which the callback returns true. The `clearCompleted` method also invokes the `invoke` method which is part of the EmberArray[97] API. `invoke` will execute a method on each object in the Array if the method exists on that object.

Reload your web browser to ensure that there are no errors and the behavior described above occurs.

Live Preview

Ember.js • TodoMVC[98]

Additional Resources

- Changes in this step in `diff` format[99]
- Handlebars Conditionals Guide[100]
- Enumerables Guide[101]

Indicating When All Todos Are Complete

Next we'll update our template to indicate when all todos have been completed. In `index.html` replace the static checkbox `<input>` with an `{{input}}`:

```
{{! ... additional lines truncated for brevity ... }}
<section id="main">
  {{outlet}}
  {{input type="checkbox" id="toggle-all" checked=allAreDone}}
</section>
{{! ... additional lines truncated for brevity ... }}
```

This checkbox will be checked when the controller property `allAreDone` is `true` and unchecked when the property `allAreDone` is `false`.

In `js/controllers/todos_controller.js` implement the matching `allAreDone` property:

[97]http://emberjs.com/api/classes/Ember.Array.html#method_invoke

[98]http://jsbin.com/ULovoJI

[99]https://github.com/emberjs/quickstart-code-sample/commit/1da450a8d693f083873a086d0d21e031ee3c129e

[100]http://emberjs.com/guides/templates/conditionals

[101]http://emberjs.com/guides/enumerables

```
// ... additional lines truncated for brevity ...
allAreDone: function(key, value) {
  return !!this.get('length') && this.isEvery('isCompleted');
}.property('@each.isCompleted')
// ... additional lines truncated for brevity ...
```

This property will be `true` if the controller has any todos and every todo's `isCompleted` property is true. If the `isCompleted` property of any todo changes, this property will be recomputed. If the return value has changed, sections of the template that need to update will be automatically updated for us.

Reload your web browser to ensure that there are no errors and the behavior described above occurs.

Live Preview

Ember.js • TodoMVC[102]

Additional Resources

- Changes in this step in `diff` format[103]
- Ember.Checkbox API documentation[104]

Toggling All Todos Between Complete and Incomplete

TodoMVC allows users to toggle all existing todos into either a complete or incomplete state. It uses the same checkbox that becomes checked when all todos are completed and unchecked when one or more todos remain incomplete.

To implement this behavior update the `allAreDone` property in `js/controllers/todos_controller.js` to handle both getting and setting behavior:

[102]http://jsbin.com/IcItARE

[103]https://github.com/emberjs/quickstart-code-sample/commit/9bf8a430bc4afb06f31be55f63f1d9806e6ab01c

[104]http://emberjs.com/api/classes/Ember.Checkbox.html

```
// ... additional lines truncated for brevity ...
allAreDone: function(key, value) {
  if (value === undefined) {
    return !!this.get('length') && this.isEvery('isCompleted', true);
  } else {
    this.setEach('isCompleted', value);
    this.invoke('save');
    return value;
  }
}.property('@each.isCompleted')
// ... additional lines truncated for brevity ...
```

If no `value` argument is passed this property is being used to populate the current value of the checkbox. If a `value` is passed it indicates the checkbox was used by a user and we should set the `isCompleted` property of each todo to this new value.

The count of remaining todos and completed todos used elsewhere in the template automatically re-render for us if necessary.

Reload your web browser to ensure that there are no errors and the behavior described above occurs.

Live Preview

Ember.js • TodoMVC[105]

Additional Resources

- Changes in this step in `diff` format[106]
- Ember.Checkbox API documentation[107]
- Computed Properties Guide[108]

Replacing the Fixture Adapter with Another Adapter

Finally we'll replace our fixture data with real persistence so todos will remain between application loads by replacing the fixture adapter with a `localstorage`-aware adapter instead.

Change `js/application.js` to:

[105] http://jsbin.com/AViZATE
[106] https://github.com/emberjs/quickstart-code-sample/commit/47b289bb9f669edaa39abd971f5e884142988663
[107] http://emberjs.com/api/classes/Ember.Checkbox.html
[108] http://emberjs.com/guides/object-model/computed-properties/

```
window.Todos = Ember.Application.create();

Todos.ApplicationAdapter = DS.LSAdapter.extend({
  namespace: 'todos-emberjs'
});
```

The local storage adapter, written by Ryan Florence, can be downloaded from its source[109]. Add it to your project as `js/libs/localstorage_adapter.js`. You may place this file anywhere you like (even just putting all code into the same file), but this guide will assume you have created the file and named it as indicated.

In `index.html` include `js/libs/localstorage_adapter.js` as a dependency:

```
<!--- ... additional lines truncated for brevity ... -->
<script src="js/libs/ember-data.js"></script>
<script src="js/libs/localstorage_adapter.js"></script>
<script src="js/application.js"></script>
 <!--- ... additional lines truncated for brevity ... -->
```

Reload your application. Todos you manage will now persist after the application has been closed.

Live Preview

Ember.js • TodoMVC[110]

Additional Resources

- Changes in this step in `diff` format[111]
- LocalStorage Adapter on GitHub[112]

[109] https://raw.github.com/rpflorence/ember-localstorage-adapter/master/localstorage_adapter.js
[110] http://jsbin.com/aZIXaYo
[111] https://github.com/emberjs/quickstart-code-sample/commit/81801d87da42d0c83685ff946c46de68589ce38f
[112] https://github.com/rpflorence/ember-localstorage-adapter

Getting Ember

Ember Builds

The Ember Release Management Team maintains a variety of ways to get Ember and Ember Data builds.

Channels

The latest Release[113], Beta[114], and Canary[115] builds of Ember and Ember data can be found here[116]. For each channel a development, minified, and production version is available. For more on the different channels read the Post 1.0 Release Cycle[117] blog post.

Tagged Releases

Past release and beta builds of Ember and Ember Data are available at Tagged Releases[118]. These builds can be useful to track down regressions in your application, but it is recommended to use the latest stable release in production.

Bower

Bower is a package manager for the web. Bower makes it easy to manage dependencies in your application including Ember and Ember Data. To learn more about Bower visit http://bower.io/[119].

Adding Ember to your application with Bower is easy; simply run `bower install ember --save`. For Ember Data, run `bower install ember-data --save`. You can also add `ember` or `ember-data` to your `bower.json` file as follows.

[113]http://emberjs.com/builds#/release
[114]http://emberjs.com/builds#/beta
[115]http://emberjs.com/builds#/canary
[116]http://emberjs.com/builds
[117]http://emberjs.com/blog/2013/09/06/new-ember-release-process.html
[118]http://emberjs.com/builds#/tagged
[119]http://bower.io/

```
{
        "name": "your-app",
        "dependencies": {
                "ember": "~1.6",
                "ember-data": "~1.0.0-beta.8"
        }
}
```

RubyGems

If your application uses a Ruby based build system, you can use the ember-source[120] and ember-data-source[121] RubyGems to access ember and ember data sources from Ruby.

If your application is built in Rails, the ember-rails[122] RubyGem makes it easy to integrate Ember into your Ruby on Rails application.

[120]http://rubygems.org/gems/ember-source
[121]http://rubygems.org/gems/ember-data-source
[122]http://rubygems.org/gems/ember-rails

Contributing To Ember.js

Adding New Features

In general, new feature development should be done on master.

Bugfixes should not introduce new APIs or break existing APIs, and do not need feature flags.

Features can introduce new APIs, and need feature flags. They should not be applied to the release or beta branches, since SemVer requires bumping the minor version to introduce new features.

Security fixes should not introduce new APIs, but may, if strictly necessary, break existing APIs. Such breakages should be as limited as possible.

Bug Fixes

Urgent Bug Fixes

Urgent bugfixes are bugfixes that need to be applied to the existing release branch. If possible, they should be made on master and prefixed with [BUGFIX release].

Beta Bug Fixes

Beta bugfixes are bugfixes that need to be applied to the beta branch. If possible, they should be made on master and tagged with [BUGFIX beta].

Security Fixes

Security fixes need to be applied to the beta branch, the current release branch, and the previous tag. If possible, they should be made on master and tagged with [SECURITY].

Features

Features must always be wrapped in a feature flag. Tests for the feature must also be wrapped in a feature flag.

Because the build-tools will process feature-flags, flags must use precisely this format. We are choosing conditionals rather than a block form because functions change the surrounding scope and may introduce problems with early return.

```
if (Ember.FEATURES.isEnabled("feature")) {
  // implementation
}
```

Tests will always run with all features on, so make sure that any tests for the feature are passing against the current state of the feature.

Commits

Commits related to a specific feature should include a prefix like [FEATURE htmlbars]. This will allow us to quickly identify all commits for a specific feature in the future. Features will never be applied to beta or release branches. Once a beta or release branch has been cut, it contains all of the new features it will ever have.

If a feature has made it into beta or release, and you make a commit to master that fixes a bug in the feature, treat it like a bugfix as described above.

Feature Naming Conventions

```
Ember.FEATURES["<packageName>-<feature>"] // if package specific
Ember.FEATURES["container-factory-injections"]
Ember.FEATURES["htmlbars"]
```

Builds

The Canary build, which is based off master, will include all features, guarded by the conditionals in the original source. This means that users of the canary build can enable whatever features they want by enabling them before creating their Ember.Application.

```
Ember.FEATURES["htmlbars"] = true;
```

`features.json`

The root of the repository will contain a features.json file, which will contain a list of features that should be enabled for beta or release builds.

This file is populated when branching, and may not gain additional features after the original branch. It may remove features.

```
{
  "htmlbars": true
}
```

The build process will remove any features not included in the list, and remove the conditionals for features in the list.

Travis Testing

For a new PR:

1. Travis will test against master with all feature flags on.
2. If a commit is tagged with [BUGFIX beta], Travis will also cherry-pick the commit into beta, and run the tests on that branch. If the commit doesn't apply cleanly or the tests fail, the tests will fail.
3. If a commit is tagged with [BUGFIX release], Travis will also cherry-pick the commit into release, and run the test on that branch. If the commit doesn't apply cleanly or the tests fail, the tests will fail.

For a new commit to master:

1. Travis will run the tests as described above.
2. If the build passes, Travis will cherry-pick the commits into the appropriate branches.

The idea is that new commits should be submitted as PRs to ensure they apply cleanly, and once the merge button is pressed, Travis will apply them to the right branches.

Go/No-Go Process

Every six weeks, the core team goes through the following process.

Beta Branch

All remaining features on the beta branch are vetted for readiness. If any feature isn't ready, it is removed from features.json.

Once this is done, the beta branch is tagged and merged into release.

Master Branch

All features on the master branch are vetted for readiness. In order for a feature to be considered "ready" at this stage, it must be ready as-is with no blockers. Features are a no-go even if they are close and additional work on the beta branch would make it ready.

Because this process happens every six weeks, there will be another opportunity for a feature to make it soon enough.

Once this is done, the master branch is merged into beta. A `features.json` file is added with the features that are ready.

Beta Releases

Every week, we repeat the Go/No-Go process for the features that remain on the beta branch. Any feature that has become unready is removed from the features.json.

Once this is done, a Beta release is tagged and pushed.

Repositories

Ember is made up of several libraries. If you wish to add a feature or fix a bug please file a pull request against the appropriate repository. Be sure to check the libraries listed below before making changes in the Ember.js repository.

Main Repositories

Ember.js - The main repository for Ember.

- https://github.com/emberjs/ember.js[123]

Ember Data - A data persistence library for Ember.js.

- https://github.com/emberjs/data[124]

Ember Website - Source for http://www.emberjs.com[125] including these guides.

- https://github.com/emberjs/website[126]

[123]https://github.com/emberjs/ember.js
[124]https://github.com/emberjs/data
[125]http://www.emberjs.com
[126]https://github.com/emberjs/website

Libraries Used By Ember

These libraries are part of the Ember.js source, but development of them takes place in a seperate repository.

`packages/ember-metal/lib/vendor/backburner.js`

- **backburner.js** - Implements the Ember run loop.
- https://github.com/ebryn/backburner.js[127]

`packages/ember-routing/lib/vendor/route-recognizer.js`

- **route-recognizer.js** - A lightweight JavaScript library that matches paths against registered routes.
- https://github.com/tildeio/route-recognizer[128]

`packages/ember-routing/lib/vendor/router.js`

- **router.js** - A lightweight JavaScript library that builds on route-recognizer and rsvp to provide an API for handling routes.
- https://github.com/tildeio/router.js[129]

`packages/metamorph`

- **Metamorph.js** - Used by Ember for databinding handlebars templates
- https://github.com/tomhuda/metamorph.js[130]

`packages/rsvp`

- **RSVP.js** - Implementation of the of Promises/A+ spec used by Ember.
- https://github.com/tildeio/rsvp.js[131]

[127]https://github.com/ebryn/backburner.js
[128]https://github.com/tildeio/route-recognizer
[129]https://github.com/tildeio/router.js
[130]https://github.com/tomhuda/metamorph.js
[131]https://github.com/tildeio/rsvp.js