**TroopJS TodoMVC Example**

**An implementation of**[**TodoMVC**](https://github.com/tastejs/todomvc/)**using**[**ToopJS**](http://troopjs.com/)

**Introduction**

This project serves two purposes:

* Provide the community a well known demo application that features the style and features of TroopJS.
* Provide a step-by-step tutorial on how to write a simple TroopJS application.

**TodoMVC deviations from the specifications**

For one reason or another there are parts of the application that deviates from the [original specifications](https://github.com/tastejs/todomvc/wiki/App-Specification). We've tried to stay as true as possible, but hey - nobody's perfect. The known deviations are:

* Use double-quotes in HTML and single-quotes in JS and CSS.

We've opted to follow the guidelines of TroopJS rather than the ones from TodoMVC for exatly the same reasons they have it posted in their [code style](https://github.com/addyosmani/todomvc/blob/gh-pages/contributing.md#code-style)

We think it's best for the project if the code you write looks like the code the last developer wrote.

We believe that's a great idea, but we want our project to look like any other *TroopJS* project, so we've stuck with our code style for this application.

* Unless it conflicts with the project's best practices, your example should use bower for package management.

For this example we're using git submodules for simple dependency management. There's a separate branch where we track the version of this application that we've submitted to TodoMVC and *that* branch uses [bower](http://twitter.github.com/bower/) for dependency management.

* This checkbox toggles all the todos to the same state as itself. Make sure to clear the checked state after the the "Clear completed" button is clicked. The "Mark all as complete" checkbox should also be updated when single todo items are checked/unchecked. Eg. When all the todos are checked it should also get checked.

Since the specification does not define what this checkbox should do when only *some* of the tasks are marked as completed, we've added an indeterminate state that covers this usecase.

**Tutorial**

This part of the document gives you a step-by-step tutorial on how the todo application was written.

**Directory structure**

Before we look at any code we'll take you through the (recommended) directory structure for a TroopJS application.

.

├── dist

├── css

├── js

│ └── lib

└── test

As you and see all application sources are contained in a top js folder. In the test you'll find test, and thedist folder we'll get the build output (note that the dist folder should be created by a build tool and ignored from source control).

Inside the js folder there's a folder called lib. This is where external *application* libraries should be stored. External libraries should be [AMD](https://github.com/amdjs/amdjs-api/wiki/AMD) compliant.

TroopJS makes use of git submodules to manage external libraries. For instructions on how submodules work you can take a look at the [documentation](http://book.git-scm.com/5_submodules.html).

**Bootstrap**

As previously noted the application resources are all contained in the js folder. In this folder there are a couple of *standard* folders that most applications would need

├── js

│ ├── lib

│ └── widget

├── css

└── img

It's also recommended that there's a index.html (the application landing-page).

So before we start we'll create a skeleton structure and add the external libraries needed for TroopJS to function.

After this is done the directory structure will look something like this (folders marked with \* are git sub-modules)

├── bower\_components

│ └── todomvc-common (\*)

├── css

├── js

│ ├── lib

│ │ ├── jquery (\*)

│ │ ├── requirejs (\*)

│ │ └── troopjs-bundle (\*)

│ └── widget

└── img

So now we can start with our todo application. The first thing we should do is to copy the [template](https://github.com/tastejs/todomvc/tree/gh-pages/template) resources to the correct locations. Once we're done with this we'll take a look at index.html

**<!doctype html>**

<html lang="en">

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">

<title>TroopJS • TodoMVC</title>

<link rel="stylesheet" href="bower\_components/todomvc-common/base.css">

*<!-- CSS overrides - remove if you don't need it -->*

<link rel="stylesheet" href="css/app.css">

</head>

<body>

<section id="todoapp">

<header id="header">

<h1>todos</h1>

<input id="new-todo" placeholder="What needs to be done?" autofocus>

</header>

*<!-- This section should be hidden by default and shown when there are todos -->*

<section id="main">

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

<label for="toggle-all">Mark all as complete</label>

<ul id="todo-list">

*<!-- These are here just to show the structure of the list items -->*

*<!-- List items should get the class `editing` when editing and `completed` when marked as completed -->*

<li class="completed">

<div class="view">

<input class="toggle" type="checkbox" checked>

<label>Create a TodoMVC template</label>

<button class="destroy"></button>

</div>

<input class="edit" value="Create a TodoMVC template">

</li>

<li>

<div class="view">

<input class="toggle" type="checkbox">

<label>Rule the web</label>

<button class="destroy"></button>

</div>

<input class="edit" value="Rule the web">

</li>

</ul>

</section>

*<!-- This footer should hidden by default and shown when there are todos -->*

<footer id="footer">

*<!-- This should be `0 items left` by default -->*

<span id="todo-count"><strong>1</strong> item left</span>

*<!-- Remove this if you don't implement routing -->*

<ul id="filters">

<li>

<a class="selected" href="#/">All</a>

</li>

<li>

<a href="#/active">Active</a>

</li>

<li>

<a href="#/completed">Completed</a>

</li>

</ul>

*<!-- Hidden if no completed items are left ↓ -->*

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

</footer>

</section>

<footer id="info">

<p>Double-click to edit a todo</p>

*<!-- Remove the below line ↓ -->*

<p>Template by <a href="http://github.com/sindresorhus">Sindre Sorhus</a></p>

*<!-- Change this out with your name and url ↓ -->*

<p>Created by <a href="http://todomvc.com">you</a></p>

<p>Part of <a href="http://todomvc.com">TodoMVC</a></p>

</footer>

*<!-- Scripts here. Don't remove this ↓ -->*

<script src="bower\_components/todomvc-common/base.js"></script>

<script src="js/app.js"></script>

</body>

</html>

TroopJS uses [RequireJS](http://requirejs.org/) for its dependency management. The recommended way to bootstrap a RequireJS application is described [here](http://requirejs.org/docs/start.html#add), but we're using an alternative way to configure RequireJS described [here](http://requirejs.org/docs/api.html#config) whereby we define the config as the global variable require **before** require.js is loaded.

Let's add the configuration (inside a script tag right before the script that includeds js/app)

"use strict";

**var** require **=** {

"baseUrl" **:** "js",

"packages" **:** [{

"name" **:** "jquery",

"location" **:** "lib/jquery",

"main" **:** "dist/jquery"

}, {

"name" **:** "poly",

"location" **:** "lib/troopjs-bundle/src/lib/poly",

"main" **:** "poly"

}, {

"name" **:** "when",

"location" **:** "lib/troopjs-bundle/src/lib/when",

"main" **:** "when"

}, {

"name" **:** "troopjs-core",

"location" **:** "lib/troopjs-bundle/src/lib/troopjs-core/src"

}, {

"name" **:** "troopjs-browser",

"location" **:** "lib/troopjs-bundle/src/lib/troopjs-browser/src"

}, {

"name" **:** "troopjs-data",

"location" **:** "lib/troopjs-bundle/src/lib/troopjs-data/src"

}, {

"name" **:** "troopjs-jquery",

"location" **:** "lib/troopjs-bundle/src/lib/troopjs-jquery/src"

}, {

"name" **:** "troopjs-requirejs",

"location" **:** "lib/troopjs-bundle/src/lib/troopjs-requirejs/src"

}, {

"name" **:** "troopjs-utils",

"location" **:** "lib/troopjs-bundle/src/lib/troopjs-utils/src"

}, {

"name" **:** "troopjs-bundle",

"location" **:** "lib/troopjs-bundle",

"main" **:** "build/maxi"

}, {

"name" **:** "troopjs-todos",

"location" **:** ".",

"main" **:** "application.min"

}],

"map" **:** {

"\*" **:** {

"template" **:** "troopjs-requirejs/template"

}

},

"deps" **:** [ "require", "jquery" ],

"callback" **:** **function** Boot (contextRequire, jQuery) {

contextRequire([ "troopjs-browser/application/widget", "troopjs-browser/route/widget" ], **function** Strap (Application, RouteWidget) {

jQuery(**function** ($) {

Application($("html"), "bootstrap", RouteWidget($(window), "route")).start();

});

});

}

};

Lets review

* "use strict";
* Tell the javascript interpreter that we run this in strict mode.
* **var** require **=** {
* Start configuring RequireJS

RequireJS supports a [configuration object as the as the global variable require](http://requirejs.org/docs/api.html#config).

* "baseUrl" **:** "js",
* Set the baseUrl to js.

**baseUrl**: the root path to use for all module lookups. So in the above example, "my/module"'s script tag will have a src="/another/path/my/module.js". baseUrl is not used when loading plain .js files, those strings are used as-is, so a.js and b.js will be loaded from the same directory as the HTML page that contains the above snippet.

If no baseUrl is explicitly set in the configuration, the default value will be the location of the HTML page that loads require.js. If a data-main attribute is used, that path will become the baseUrl.

The baseUrl can be a URL on a different domain as the page that will load require.js. RequireJS script loading works across domains. The only restriction is on text content loaded by text! plugins: those paths should be on the same domain as the page, at least during development. The optimization tool will inline text! plugin resources so after using the optimization tool, you can use resources that reference text! plugin resources from another domain.

* "packages" **:** [{
* "name" **:** "jquery",
* "location" **:** "lib/jquery",
* "main" **:** "dist/jquery"
* }, {
* "name" **:** "poly",
* "location" **:** "lib/troopjs-bundle/src/lib/poly",
* "main" **:** "poly"
* }, {
* "name" **:** "when",
* "location" **:** "lib/troopjs-bundle/src/lib/when",
* "main" **:** "when"
* }, {
* "name" **:** "troopjs-core",
* "location" **:** "lib/troopjs-bundle/src/lib/troopjs-core/src"
* }, {
* "name" **:** "troopjs-browser",
* "location" **:** "lib/troopjs-bundle/src/lib/troopjs-browser/src"
* }, {
* "name" **:** "troopjs-data",
* "location" **:** "lib/troopjs-bundle/src/lib/troopjs-data/src"
* }, {
* "name" **:** "troopjs-jquery",
* "location" **:** "lib/troopjs-bundle/src/lib/troopjs-jquery/src"
* }, {
* "name" **:** "troopjs-requirejs",
* "location" **:** "lib/troopjs-bundle/src/lib/troopjs-requirejs/src"
* }, {
* "name" **:** "troopjs-utils",
* "location" **:** "lib/troopjs-bundle/src/lib/troopjs-utils/src"
* }, {
* "name" **:** "troopjs-bundle",
* "location" **:** "lib/troopjs-bundle",
* "main" **:** "build/maxi"
* }, {
* "name" **:** "troopjs-todos",
* "location" **:** ".",
* "main" **:** "application.min"
* }],
* Configures loading modules from CommonJS packages.

**packages**: RequireJS supports loading modules that are in a [CommonJS Packages directory](http://wiki.commonjs.org/wiki/Packages/1.1)structure, but some additional configuration needs to be specified for it to work. Specifically, there is support for the following CommonJS Packages features:

* + A package can be associated with a module name/prefix.
  + The package config can specify the following properties for a specific package:
    - **name**: The name of the package (used for the module name/prefix mapping)
    - **location**: The location on disk. Locations are relative to the baseUrl configuration value, unless they contain a protocol or start with a front slash (/).
    - **main**: The name of the module inside the package that should be used when someone does a require for "packageName". The default value is "main", so only specify it if it differs from the default. The value is relative to the package folder.

There's further information available in the RequireJS documentation about [Loading Modules from Packages](http://requirejs.org/docs/api.html#packages).

* "deps"**:** [ "require", "jquery" ]
* Depend on require and jquery

**deps**: An array of dependencies to load. This is useful when require is defined as a config object before require.js is loaded, and you want to specify dependencies to load as soon as require() is defined.

* "callback" **:** **function** Boot (contextRequire, jQuery) {
* The callback that will be called after **deps** have been resolved (in this case called Boot).

A function to execute after **deps** have been loaded. Useful when require is defined as a config object before require.js is loaded, and you want to specify a function to require after the configuration's **deps**array has been loaded.

* contextRequire([ "troopjs-browser/application/widget", "troopjs-browser/route/widget" ], **function** Strap(Application, RouteWidget) {
* Use the context require to load troopjs-browser/application/widget andtroopjs-browser/route/widget, and once that is completed call the Strap function.
* jQuery(document).ready(**function** ($) {
* Add a standard ready handler to the document
* Application($("html"), "bootstrap", RouteWidget($(window), "route")).start();
* Create and attach the bootstrap application to $("html") and add a RouteWidget attached to$(window) as a child. Then start the application.

Now we've configure our application to use RequireJS and set up the application entry point.

**Adding some widgets**

Lets go back and look at [index.html](https://github.com/troopjs/troopjs-todos/blob/master/index.html). We want to try to break out functionality into small (somewhat self-contained) widgets, and the natural place to start is adding and displaying todo items.

There are three main classes of modules in TroopJS

* components are the base building block of anything TroopJS
* gadgets extend components with methods like publish and subscribe
* widgets extend gadgets with UI related methods like html and after

Let's do this by adding *weave* instructions in the HTML using data-weave attributes.

* html <input id="new-todo" placeholder="What needs to be done?" autofocus data-weave="widget/create">
* html <ul id="todo-list" data-weave="widget/list">

TroopJS *weaves* widgets to the DOM by traversing it and finding elements that have a data-weaveattribute. When weaving an element TroopJS will:

* Locate (and if needed async load) the module containing the widget
* Instantiate the widget and attach the jQuery wrapped DOM element to the created instance
* Wire the instance (basically reflect on the instance and scan for well-known method signatures), more on this later

If you look at the modified [index.html](https://github.com/troopjs/troopjs-todos/blob/master/index.html) you can locate all the widgets simply by searching for the data-weaveattribute on any element.

**[Create widget](https://github.com/troopjs/troopjs-todos/blob/master/js/widget/create.js)**

The first widget to deal with is the create widget

Widgets are named after where they are located (relative to baseUrl) in the source tree. A general rule is to simply add .js to the widget name to locate the file, so widget/create can be found injs/widget/create.js

define([ "troopjs-browser/component/widget" ], **function** CreateModule(Widget) {

"use strict";

**var** ENTER\_KEY **=** 13;

**return** Widget.extend({

"dom/keyup"**:** **function** onKeyUp($event) {

**var** me **=** **this**;

**var** $element **=** me.$element;

**var** value;

**if** ($event.keyCode **===** ENTER\_KEY) {

value **=** $element.val().trim();

**if** (value **!==** "") {

me.publish("todos/add", value)

.then(**function** () {

$element.val("");

});

}

}

}

});

});

Let's go through this widget

* define([ "troopjs-browser/component/widget" ], **function** CreateModule(Widget) {
* Start the definition of this module and declare its dependencies. The module is (internally) namedCreateModule and it depends on troopjs-browser/component/widget which will be available inside the module as Widget

If you look above in index.html you'll find a package definition for troopjs-browser that points tolib/troopjs-bundle/src/lib/troopjs-browser/src. This means that troopjs-browser/... actually resolves to lib/troopjs-bundle/src/lib/troopjs-browser/src/...

* "use strict";
* Be strict.
* **var** ENTER\_KEY **=** 13;
* Declare a constant for ENTER\_KEY corresponding to the keyCode of enter.
* **return** Widget.extend({
* The result of this module is extending Widget
* "dom/keyup" **:** **function** onKeyUp($event) {
* This is where wiring becomes important. As mentioned above, wiring scans for well-known method signatures, and dom/\* is one of these. In this instance, we're indicating that we want to add a handler for the DOM keyup event.

For DOM handlers, the first argument is the original jQuery event object.

* **var** me **=** **this**;
* **var** $element **=** me.$element;
* **var** value;
* **if** ($event.keyCode **===** ENTER\_KEY) {
* value **=** $element.val().trim();
* **if** (value **!==** "") {
* me.publish("todos/add", value)
* .then(**function** () {
* $element.val("");
* });
* }
* }
  + Save this as me so we can use it inside of closures
  + Save me.$element (woven element) as $element
  + Check if the keyCode of the event was enter
    - Store the trimmed value of the element as value
    - publish value on todos/add
    - Once all handlers are completed, reset $element.

**[Count widget](https://github.com/troopjs/troopjs-todos/blob/master/js/widget/count.js)**

Next we'll take a look at the count widget. This widget shows a counter that informs the user of how many active items are in the list.

define([ "troopjs-browser/component/widget", "poly/array" ], **function** CountModule(Widget) {

"use strict";

**function** filter(item) {

**return** item **!==** **null** **&&** **!**item.completed;

}

**return** Widget.extend({

"hub:memory/todos/change" **:** **function** onChange(items) {

**var** count **=** items.filter(filter).length;

**this**.$element.html("<strong>" **+** count **+** "</strong> " **+** (count **===** 1 **?** "item" **:** "items") **+** " left");

}

});

});

Let's look at what new things we can find.

* **function** filter(item) {
* **return** item **!==** **null** **&&** **!**item.completed;
* }
* A static filter later used by $.grep to count active items.
* "hub:memory/todos/change" **:** **function** onChange(items) {
* Again with the well-known signatures. This signature tells TroopJS that we want to add a subscription to thetodos/change topic, *and* that if a previous value was published on this topic *before* we added our subscription, we'd like to get a callback with that value (this is what :memory adds to the mix).
* **var** count **=** items.filter(filter).length;
* This filters the list to only contain active items. After that we count the number of items in the array and store as count.
* **this**.$element.html("<strong>" **+** count **+** "</strong> " **+** (count **===** 1 **?** "item" **:** "items") **+** " left");
* Update the $element HTML with a pluralized (if needed) text indicating what the current count is.

**[Clear widget](https://github.com/troopjs/troopjs-todos/blob/master/js/widget/clear.js)**

The clear widget is quite similar to the count widget, but the opposite. Instead of counting the number of active items in the list, it counts the number of completed items in the list.

define([ "troopjs-browser/component/widget", "poly/array" ], **function** ClearModule(Widget) {

"use strict";

**function** filter(item) {

**return** item **!==** **null** **&&** item.completed;

}

**return** Widget.extend({

"hub:memory/todos/change" **:** **function** onChange(items) {

**var** count **=** items.filter(filter).length;

**this**.$element.text("Clear completed (" **+** count **+** ")").toggle(count **>** 0);

},

"dom/click" **:** **function** onClear() {

**this**.publish("todos/clear");

}

});

});

What looks different here?

* **function** filter(item) {
* **return** item **!==** **null** **&&** item.completed;
* }
* Almost the same filter as before, but this time for completed items.
* **this**.$element.text("Clear completed (" **+** count **+** ")").toggle(count **>** 0);
* Update the $element HTML with a pluralized (if needed) text indicating what the current count istoggle.
* "dom/click" **:** **function** onClear() {
* **this**.publish("todos/clear");
* }
* Register a click handler that will publish todos/clear on the pubsub every time it is invoked.

**[Mark widget](https://github.com/troopjs/troopjs-todos/blob/master/js/widget/mark.js)**

The mark widget can do two things

* It allows the user to mark all the items as either completed or active with one click
* It shows the aggregate status of all the items in the list
  + **Unchecked** if *no* items are completed
  + **Checked** if *all* items are completed
  + **Indedeterminate** if *some* items are completed

define([ "troopjs-browser/component/widget", "jquery", "poly/array" ], **function** MarkModule(Widget, $) {

"use strict";

**return** Widget.extend({

"hub:memory/todos/change" **:** **function** onChange(items) {

**var** total **=** 0;

**var** completed **=** 0;

**var** $element **=** **this**.$element;

items.forEach(**function** (item) {

**if** (item **===** **null**) {

**return**;

}

**if** (item.completed) {

completed**++**;

}

total**++**;

});

**if** (completed **===** 0) {

$element

.prop("indeterminate", **false**)

.prop("checked", **false**);

}

**else** **if** (completed **===** total) {

$element

.prop("indeterminate", **false**)

.prop("checked", **true**);

}

**else** {

$element

.prop("indeterminate", **true**)

.prop("checked", **false**);

}

},

"dom/change" **:** **function** onMark($event) {

**this**.publish("todos/mark", $($event.target).prop("checked"));

}

});

});

Let's start with the first item, showing an aggregate status

* "hub:memory/todos/change" **:** **function** onChange(items) {
* First register a handler for todos/change. You should recognize this by now as any widget interesting in changes of the list have handlers registered for this topic.
* **var** total **=** 0;
* **var** completed **=** 0;
* **var** $element **=** **this**.$element;
* items.forEach(**function** (item) {
* **if** (item **===** **null**) {
* **return**;
* }
* **if** (item.completed) {
* completed**++**;
* }
* total**++**;
* });
* Iterate items to determine how many non null items are there in total and how many of them arecompleted.
* **if** (completed **===** 0) {
* $element
* .prop("indeterminate", **false**)
* .prop("checked", **false**);
* }
* **else** **if** (completed **===** total) {
* $element
* .prop("indeterminate", **false**)
* .prop("checked", **true**);
* }
* **else** {
* $element
* .prop("indeterminate", **true**)
* .prop("checked", **false**);
* }
* Update the $element indeterminate and checked properties to reflect the result.

And then the second item - batch interaction:

* "dom/change" **:** **function** onMark($event) {
* **this**.publish("todos/mark", $($event.target).prop("checked"));
* }
* Register a change handler that will publish todos/mark on the pubsub with the current checked status of the checkbox as an argument.

**[Filters widget](https://github.com/troopjs/troopjs-todos/blob/master/js/widget/filters.js)**

The filters widget reflects the current filter status and allows the user to apply filters to the list.

define([ "troopjs-browser/component/widget", "jquery" ], **function** FiltersModule(Widget, $) {

"use strict";

**return** Widget.extend({

"hub:memory/route" **:** **function** onRoute(uri) {

**this**.publish("todos/filter", uri.source);

},

"hub:memory/todos/filter" **:** **function** onFilter(filter) {

$("a[href^='#']")

.removeClass("selected")

.filter("[href='#" **+** (filter **||** "/") **+** "']")

.addClass("selected");

}

});

});

Let's take a closer look

* "hub:memory/route" **:** **function** onRoute(uri) {
* **this**.publish("todos/filter", uri.source);
* },
* This will register a handler for the route topic that will republish uri.source on todos/filter. Theroute topic is published on each time the route (anything after # in the url) changes. The uri object is a parsed version of the route. Also note the :memory part that ensures we always get the latest value published on this topic.
* "hub:memory/todos/filter" **:** **function** onFilter(filter) {
* $("a[href^='#']")
* .removeClass("selected")
* .filter("[href='#" **+** (filter **||** "/") **+** "']")
* .addClass("selected");
* }
* Registers a handler for the todos/filter topic (that we publish above). Sets the default filter (if none is provided) to / then finds all child elements matching the css selector a[href^='#'] (an anchor element where the href attribute starts with #) then either add or remove the selected css class (depending on if the filter matches).

**[Display widget](https://github.com/troopjs/troopjs-todos/blob/master/js/widget/display.js)**

The display widget shows or hides its contents depending on the status of the list

define([ "troopjs-browser/component/widget", "poly/array" ], **function** DisplayModule(Widget) {

"use strict";

**function** filter(item) {

**return** item **!==** **null**;

}

**return** Widget.extend({

"hub:memory/todos/change"**:** **function** onChange(items) {

**this**.$element.toggle(items.some(filter));

}

});

});

Quite simply it registers a handler for todos/change that will toggle depending on the result of items.some.

**[List widget](https://github.com/troopjs/troopjs-todos/blob/master/js/widget/list.js)**

The list widget is where all the magic happens. It is by far the largest widget and it contains all the logic that deals with the list.