Creating Your First Lit-Element

## Step 0 – Setting up the project

Setup the D2L lit-element-template from <https://github.com/BrightspaceUI/lit-element-template>

This involves creating a new repository from the template and cloning it to your local machine. After cloning to your local machine run

Node setup/configure-repo.js and give it a name like todo-list

Next we should install node components by running:

Npm install

Now we will install the polymer cli (command line interface) by running:

Npm install -g polymer-cli

Finally we can start running our demo (of the template):

Polymer serve

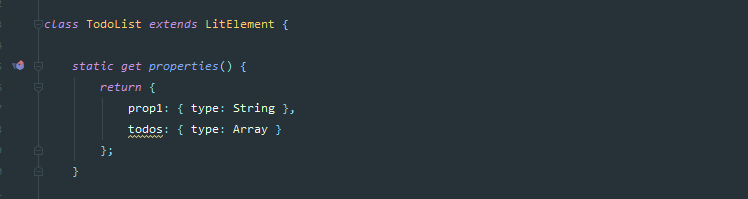
Click on your reusable component link and you should see



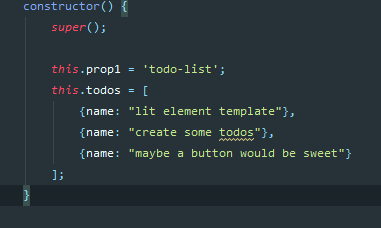
## Step 1 Making a todo list

Lets modify `todo-list.js` to add and display todos.

Lit Element has a static getter called *properties* that returns an object that defines the property names and types. Add in an entry called todos that is of type Array.



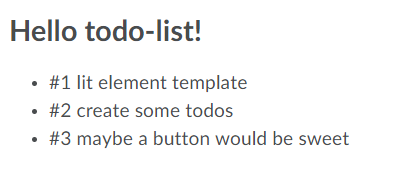
Now Lets create some sample todos in the constructor, create some sample todos in the constructor by doing:



Now to see our todos we need to add them to the template which is defined in the render method. We can add to the sample html provided here. Lets create an unordered list html element <ul> and then we will use lit element to create a map of the todos and populate list item elements. We can use ${javascript values} in lit element. So for our todos we could do ${this.todos.map( function to return html)} inside the template. Try to see if you can create this, or take a peek below for the solution.



So now we have added new properties, set those properties in constructor and used the properties in the render method. Now we have our todos in a list.



## Step 2 – Adding todos by binding to events

Lets add some more html elements to let us add todos using a text field and a button. Then we will bind a function to the button click event of the button.

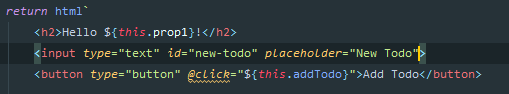
Inside the render method, above our ul element, add an input element of type text and give the input element an id of “new-todo” and some placeholder text.

Also add a button element of type “button” with the text “Add Todo”.

Now we can bind a method to the click event of the button Bind a method to the click event by adding:

@click = “${this.addTodo}”

To the button attribute.



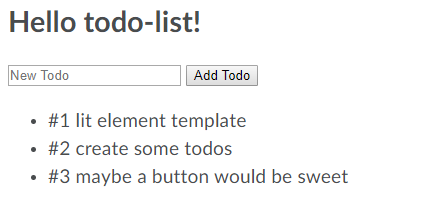
Now lets define the addTodo method, create a new method in your class called addTodo. We can access the DOM directly using the “new-todo” id we defined above. The dom is access through the this.shadowRoot call. Fetch the value in the text box by calling:

*const* newTodo = *this*.shadowRoot.getElementById('new-todo').value;

Now lets add this todo to our list and at the same time make a new array object. The new array object will cause lit to rerender the todos, changing properties of an object will not cause a rerender but a rerender can be requested by calling this.requestUpdate();

*this*.todos = [  
 ...*this*.todos,  
 { name: newTodo }  
];

Now our todo app should like this and you should be able to add todos as well



## Step 3 - Creating a component to be used by other components.

In this step we are going to create a todo component that we can use in our todo list. Lets create a file called todo.js.

Like with our todo-list component we need to import from lit-element.js

*import* { css, html, LitElement } *from* 'lit-element/lit-element.js';

Now make a class called Todo that extends LitElement:

* Add in a static get properties method with the property name which is a string.
* Add a render method to return html that renders our name.
* Add in the customElements.define( name, Todo) after the class, name could be d2l-labs-todo, remember that it has to have at least one ‘-‘ character.



Ok now we have to go back todo-list and import this new component. Add to the imports

*import* './todo.js';

Now go to the render function of todo-list and render todo components in our list

<d2l-labs-todo name="${todo.name}"></d2l-labs-todo>

Passing in the name attriutes sets the property here.

## Step 4 – Firing and listening for events

Now that we have our list of todos and a way to add new todos, lets add the ability to delete todos. On our todo component lets add a property for an id and a delete button to the html.

id: {type: Number}

<button type="button" @click="${*this*.deleteTodo}">Delete</button>

Now we need to define a deleteTodo method and fire a custom event from there, you can read more about events here <https://lit-element.polymer-project.org/guide/events>

Try and define a custom event or go ahead and take a look at the code here:

deleteTodo() {  
 *let* deleteEvent = *new* CustomEvent( 'delete-todo', {  
 detail: {  
 message: 'delete button clicked',  
 id: *this*.id  
 },  
 bubbles:*true*,  
 composed: *true* });  
 *this*.dispatchEvent(deleteEvent);  
}

Now we need todo-list to listen for this event and remove the todo since it maintains the list of todos. Add an event listener in the constructor of todo-list.

*this*.addEventListener( 'delete-todo', *this*.handleDeleteTodo);

Now create a method called handleDeleteTodo that has an event as the parameter. The index of the todo in the array can be pulled from the event like this:

*let* deletedId = event.detail.id;

Now we want to remove that index from the array by splicing the array.

*this*.todos.splice(deletedId, 1);

Finally because we modified the array instead of replacing the array we need to tell lit element to rerender:

*this*.requestUpdate();

Populate the id of the todo by using the index of the for loop that renders the todos:

<d2l-labs-todo name="${todo.name}" id="${index}">

## Step 5 – Slots

We can render things to our children element by using a slot in the child’s template. The html tag is the <slot></slot> tag.

Lets display an icon for our todos but we will use the slot tag to do it so that the list can specify the icon to be rendered.

Add a slot tag to our todo render method at the top.

Now in lets pick out an icon from BrightspaceUI’s core <https://github.com/BrightspaceUI/core>

Install by running in your terminal

npm install @brightspace-ui/core

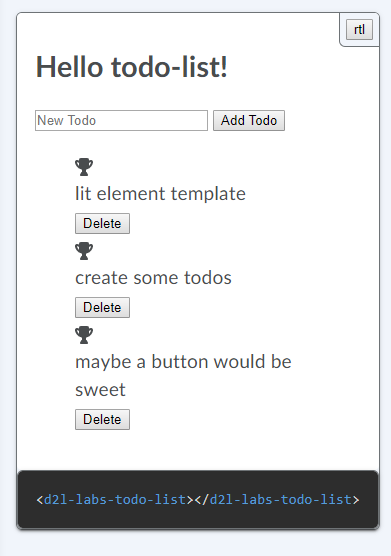
Now in your todo-list.js import the core icons

*import* '@brightspace-ui/core/components/icons/icon.js';

Now add a d2l-icon in between our d2l-labls-todo tags:

<div>  
 <d2l-labs-todo name="${todo.name}">  
 <d2l-icon icon="tier1:awards"></d2l-icon>  
 </d2l-labs-todo>  
 </div>

Great now we are all done, we created a lit element component that has properties, events, slots and child elements!



## Step 6 - Style

We can add in some css to our todo.js to get all of the elements onto the same line.

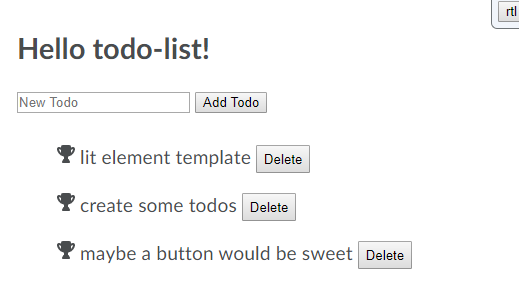
Add in a static get styles() method and return a block of css.

*static get styles*() {  
 *return* css`  
 .todo-wrapper {  
 display: flex;  
 padding-top: 0.5rem;  
 padding-bottom: 0.5rem;  
 }  
 .todo {  
 padding-left: 5px;  
 padding-right: 5px;  
 }  
 `;  
}

Now in our html render method we can add in those classes:

render() {  
 *return* html`  
 <div class="todo-wrapper">  
 <slot></slot>  
 <div class="todo">${*this*.name}</div>  
 <button type="button" @click="${*this*.deleteTodo}">Delete</button>  
 </div>  
 `  
}

and now we have all our todo on one line in the list.



That’s all for the tutorial! Thank you for following along!

## What’s Next?

* You could add some more style to match what the Brightspace look and feel, use things like d2l-button instead of button.
* Have the todo names be editable
* Add in a completed checkmark