Connecting Form Components

Thus far, our application has been non-interactive. We can currently click on Pilot and Mech list items to select them, but there's no way to modify anything. It's time to start implementing some interactivity.

Creating the Form Update Logic

Our first task is to hook up the `<UnitInfo>` form so that we can edit the current unit's name and change what Battletech House or mercenary group they're affiliated with. We'll need to add an action type and a case reducer to handle those updates, then modify `<UnitInfo>` so that it dispatches the action in response to `onChange` callbacks from the inputs.

> **Commit 3360ece: Implement initial unit info update handling**

The action/reducer changes are simple. New action type, a matching action creator, and a case reducer:

features/unitInfo/unitInfoReducer.js

```
+import _ from "lodash";

import {DATA_LOADED} from "features/tools/toolConstants";
+import {UNIT_INFO_UPDATE} from "./unitInfoConstants";

+function updateUnitInfo(state, payload) {
+   const updates = _.pick(payload, ["name", "affiliation"]);
+
+   return {
+       ...state,
+       ...updates
+   };
+}
```

```
export default createReducer(initialState, {
    [DATA_LOADED] : dataLoaded,
+   [UNIT_INFO_UPDATE] : updateUnitInfo,
});
```

We *could* create entirely separate action types and reducers for updating the "Name" field and the "Affiliation" field, but that would be a waste of effort. **Defining action payloads and reducer logic involves tradeoffs, and it's up to you to decide when actions should be more specific or more general**.

I usually try to avoid reducers that just blindly copy whatever the action contains. In this case, it would be really easy to just do `return {...state, ...payload}`, but adding the `_.pick()` call makes it clearer what fields we're expecting. We could also have destructured the fields we expected and put them back into a new object.

Connecting a Controlled Input

One of the most important concepts to understand when learning React is the idea of "controlled inputs". If you're not familiar with controlled inputs, go read Gosha Arinich's article **Controlled and uncontrolled form inputs in React don't have to be complicated**, or the additional articles on forms in React linked at the end of the post.

As a quick summary, **a controlled input is an input with a `value` prop and an `onChange` handler**. That means that **the input is being *told* what its value is at all times, instead of the application asking the input element for its value when it's time to submit the form**. Managing controlled inputs does take additional work, but ultimately makes the application much easier to think about, since all the form data is already being stored by the application.

Values for controlled inputs can be stored by a React component, or passed all the way back to a Redux store. Since the `<UnitInfo>` component is already connected, we just need to pass in the action creator, add an `onChange` handler for the "Affiliation" dropdown, and dispatch the action appropriately. While we're at it, adding a change handler means it's a good time to convert `UnitInfo` from a functional component to a class component. We'll skip that conversion in this snippet for simplicity.

> **Commit 740e66: Implement initial change handling for UnitInf**

Commit 740ca66: Implement initial change handling for UnitInfo

features/unitInfo/UnitInfo.jsx

```
+import {updateUnitInfo} from "./unitInfoActions";

+const actions = {
+   updateUnitInfo,
+};

class UnitInfo extends Component {
+   onAffiliationChanged = (e, result) => {
+       const {name, value} = result;
+
+       const newValues = { [name] : value};
+       this.props.updateUnitInfo(newValues);
+   }

// Omit unrelated rendering

                        <Dropdown
+                           name="affiliation"
                            selection
                            options={FACTIONS}
                            value={affiliation}
+                           onChange={this.onAffiliationChanged}
                        />

// Omit rest of component

-export default connect(mapState)(UnitInfo);
+export default connect(mapState, actions)(UnitInfo);
```

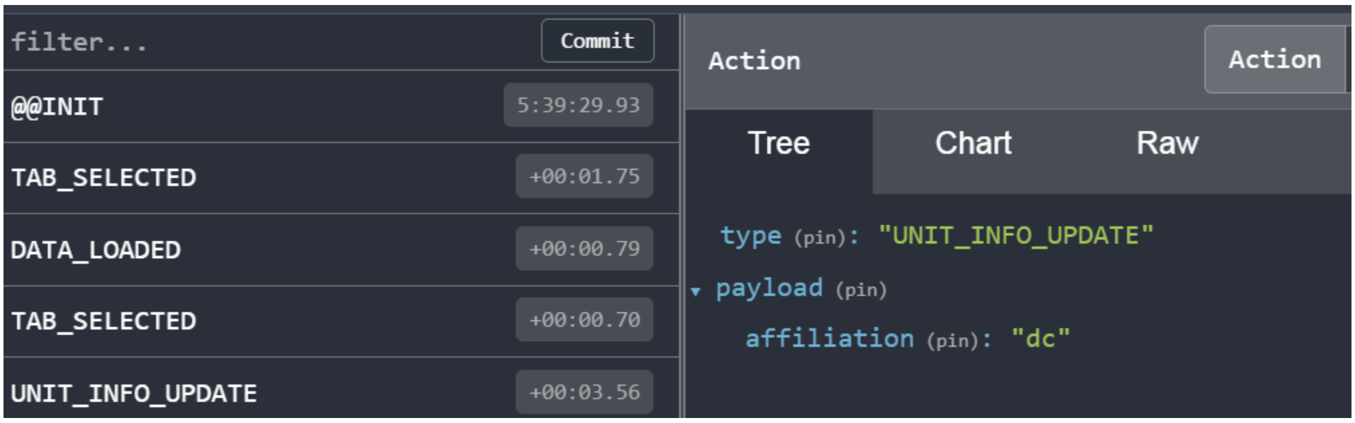
A few things to note about the `onAffiliationChanged` handler:

First, we're using the stage 3 Class Properties syntax to define an auto-bound method using an arrow function, so that `this` inside the callback correctly refers to the component instance.

Second, per Semantic-UI-React's component props documentation for `Dropdown`, the `onChange` callback is called with two parameters: the React event object, and a result object that contains the name of the component and its new value (like `{name : "affiliation", value : "wd"}`). We want to reshape that into something like `{affiliation : "wd"}`, so we use the ES6 object computed properties syntax to create the new object.

Finally, since we used the object shorthand syntax for binding up action creators with `connect()`, calling `this.props.updateUnitInfo(newValues)` immediately dispatches the action.

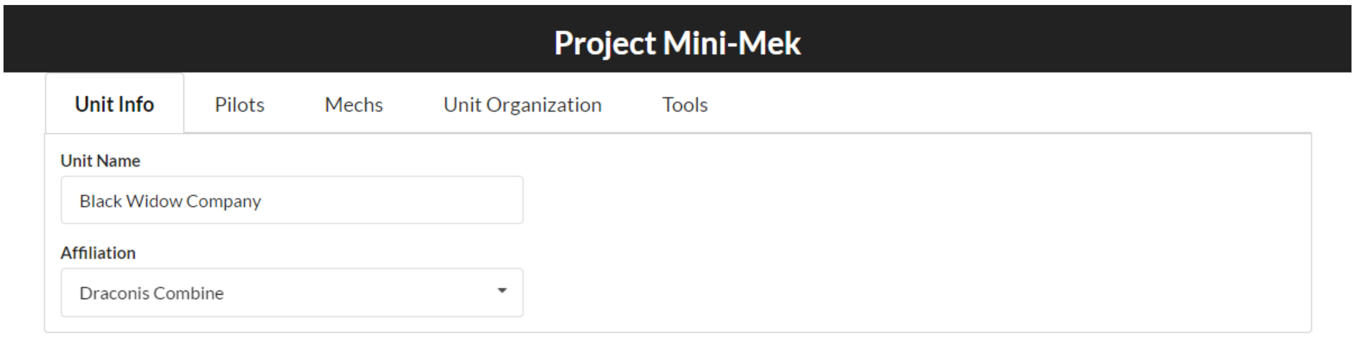
Now, if we go to the Unit Info tab and select “Draconis Combine” from the dropdown, we should see the dispatched action in our DevTools:



```
@@INIT              5:39:29.93
TAB_SELECTED        +00:01.75
DATA_LOADED         +00:00.79
TAB_SELECTED        +00:00.70
UNIT_INFO_UPDATE    +00:03.56

type (pin): "UNIT_INFO_UPDATE"
▾ payload (pin)
    affiliation (pin): "dc"
```

And the dropdown should now read “Draconis Combine”:



Unit Name: Black Widow Company

Affiliation: Draconis Combine

From there, we can enable editing the “Name” field with just another change handler:

> **Commit d008a79: Connect UnitInfo name input**

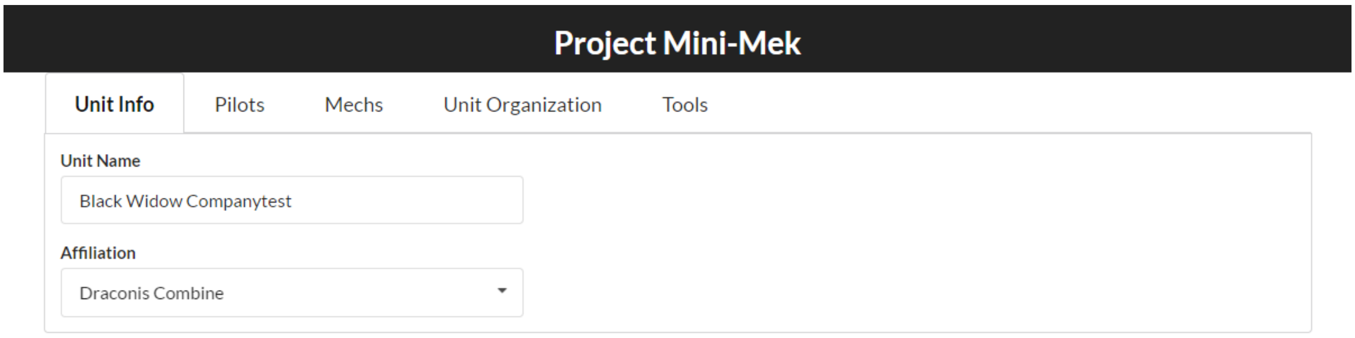
features/unitInfo/UnitInfo.jsx

```
+    onNameChanged = (e) => {
+        const {name, value} = e.target;
+
+        const newValues = { [name] : value};
+        this.props.updateUnitInfo(newValues);
+    }

// Omit other rendering

                    <Form.Field name="name" width={6}>
                        <label>Unit Name</label>
                        <input
                            placeholder="Name"
                            name="name"
                            value={name}
+                           onChange={this.onNameChanged}
                        />
                    </Form.Field>
```

And now we can happily type some gibberish into the “Unit Name” field, and see it show up:



Unit Name
Black Widow Companytest
Affiliation
Draconis Combine

It may not look like much, but this is some great progress! We can now edit the name and affiliation of our combat unit.

Retrieving Values from Input Events

Right now we’re manually extracting the `name` and `value` fields from the text input’s `onChange` event. There’s some differences in how HTML inputs structure their events. Checkboxes in particular use a different field name (`checked` instead of `value`). We can write a small utility function to extract the name and value from events, and do the object formatting for us.

> **Commit 2539a1d: Add a utility function to extract values from input events**

common/utils/clientUtils.js

```
import {isObject} from "lodash";
```

```
export function getValueFromEvent(e) {

    const {target} = e;

    let newValues;

    if(target) {
        const value = (target.type === "checkbox") ? target.checked : targ
et.value;
        newValues = {
            [target.name] : value,
        };
    }
    else if(isObject(e)) {
        newValues = e;
    }

    return newValues;
}
```

And that simplifies our code in `<UnitInfo>` a bit:

features/unitInfo/UnitInfo/UnitInfo.jsx

```
+import {getValueFromEvent} from "common/utils/clientUtils";


    onNameChanged = (e) => {
-       const {name, value} = e.target;
-
-       const newValues = { [name] : value};
+       const newValues = getValueFromEvent(e);
        this.props.updateUnitInfo(newValues);
    }
```

Type in the input, we get back a name/value object as needed, and we dispatch it. Looks great.

There *is* one problem with our text input that we need to address, but we'll deal with that in the next major section of the course.