



# Advanced Actions

---

Cool things to make actions more understandable

## tl;dr

Certain action practices may not give you additional capabilities but they can make your code cleaner:

- Action type constants
- Action type enumeration
- Action creators
- Action creator enumeration

# Action constants

---

Q: What happens if you have a reducer with an action of

```
case "SET_FIRSTNAME":  
return {...person, first: action.first}
```

But try this:

```
const first = txtBox1.value;  
store.dispatch({type: "SET_FRISTNAME", first});
```

A: Nothing happens. No errors are thrown. Try debugging that!!

Q: Now what happens if you changed the reducer like this:

```
const SET_FIRSTNAME = "SET_FIRSTNAME";  
...  
case SET_FIRSTNAME:  
  return {...person, first: action.first}
```

And try this:

```
const SET_FIRSTNAME = "SET_FIRSTNAME";  
...  
const first = txtBox1.value;  
store.dispatch({type:SET_FRISTNAME, first});
```

A: JavaScript throws during development

## So some devs create an action-types.js

```
export const SET_LATLON = "SET_LATLON";  
export const SET_PRECIP_PROBABILITY=  
"SET_PRECIP_PROBABILITY";  
export const SET_PRECIP_TYPE = "  
SET_PRECIP_TYPE";
```

- Then, everywhere one is used ...

```
import { SET_LATLON } from './action-types';  
...  
case SET_LATLON:  
  // Do stuff
```

# The action enumeration

---



Note: This technique may not be popular yet, but it is very clean

- Not a requirement, just a good idea.
- Eliminates magic strings and thus typos



# Create an action "enumeration"

- All uppercased. Underscores between the words.

```
const ActionTypes = {  
  ADD_FOO: "ADD_FOO",  
  REMOVE_FOO: "REMOVE_FOO",  
  SUBMIT_ORDER: "SUBMIT_ORDER",  
  SET_FOO: "SET_FOO",  
};  
export default ActionTypes;
```

## To use your enumeration...

```
import actionTypes from './action-types';  
...  
case actionTypes.SET_LATLON:  
  // Do stuff
```

# Action creators

---

## Actions are tough to remember also

Again, you're coding and you're trying to set the user's location.  
what does the action look like?

```
store.dispatch( /* what goes here? */) 
```

Maybe you remember the action type, but what else is in the  
payload?

## Solution: Use an action creator!

```
function setUserAction(first, last, birthdate)
{
    return {
        type: SET_USER, first, last, birthdate
    };
}
```

- Then you

```
store.dispatch(setUserAction("April",
    "Ludgate", aDate));
```

- This works because the function returns a properly formatted action.


# Action enumerations

---



Note: This is an original concept so you won't find it by searching online.

- Not that others haven't discovered it independently also.



If we use action creators, we don't have to remember the action type and payload.

But you still need to remember the name of the action creator!

**Solution: Put all of your action creators in an enumeration**



## In actions.ts

```
const setUser =  
  (first, last, birthdate) =>  
    ({type:SET_USER, first, last, birthdate});  
const setPassword =  
  (pass) =>  
    ({type:SET_PASS, pass});  
// etc., etc.  
export const actions = {  
  setUser,  
  setPassword,  
  /* etc. etc. */  
};
```

## Then to dispatch an action ...

```
import { actions } from 'actions.js';  
...  
store.dispatch(actions.
```

- ... and your IDE's intellisense kicks in so you can pick an action from the list and get prompted for the arguments to pass in.

```
store.dispatch(actions.setUser(  
    "Chris", "Trager", someDate));
```

## tl;dr

Certain action practices may not give you additional capabilities but they can make your code cleaner:

- Action type constants
- Action type enumeration
- Action creators
- Action creator enumeration