How a component and a container are wired with actions in Redux framework, and how the flow of an action proceeds

We wrote this small pamphlet to illustrate how the boilerplate for our React/Redux application is currently configured. The point here is to understand on a high level how the individual parts come together and what types have to be supplied where.
It is a small example for a fictional component called MyComponent which defines a simple "trigger" action. It is by no means comprehensive and serves to illustrate a basic but nontrivial use.

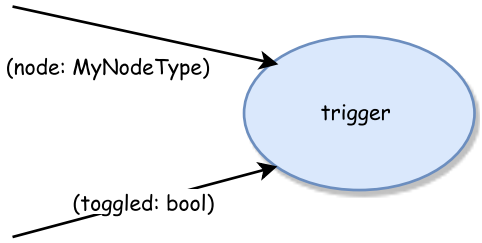
All functions which have to be declared in the view component are coloured in blue, the functions which must be in the container are coloured in green, and the action creator functions are coloured in red, and the parts for the reducer in green.

Remember that the files in the appropriate folders should have the same names to logically group them together! In this case, 'mycomponent.ts'

Phil & Tobias (May 5, 2017)

Example action: a "trigger" function taking 2 arguments, one a "node" and a bool. This is defined in your <Component>Props interface and is a void function.

```
interface MyComponentProps {
  ...
  trigger: (node: MyNodeType, toggled: boolean) => void;
}
```



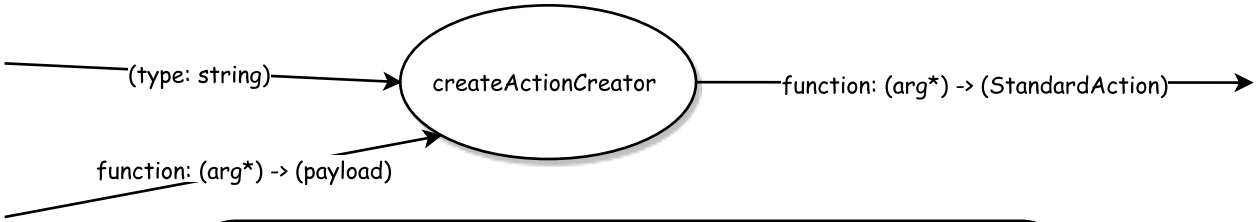
Because the component is only the presentational layer ("view"), we need to map the actual functionality in the container, which glues together model (state) and the view. Here we use the convenience "connect" functionality typical for react-redux and specify the `mapStateToProps` and `mapDispatchToProps` functions for the framework to "wire up" states and actions for our component.

```
const mapDispatchToProps = {
  trigger: triggerNode,
};
```

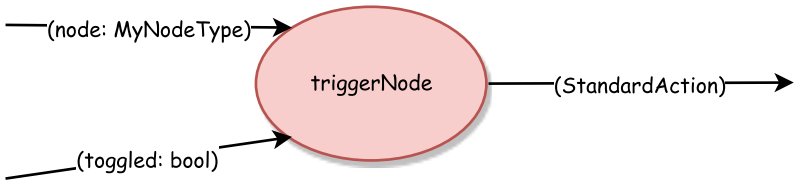
where `triggerNode` is actually imported from the corresponding action file:

```
export const TRIGGER_NODE = 'TRIGGER_NODE';
export const triggerNode =
  createActionCreator(TRIGGER_NODE,
    (node: MyNodeType, toggled: boolean) => ({node, toggled}));
```

The string `'TRIGGER_NODE'` is the action type which will be used later. `createActionCreator` takes as arguments the action type (a string) and a function that maps the "payload" arguments from the component (here a node and a boolean) to unified "payload" object, which will be part of a `StandardAction` object that is later moved around by Redux. That is:

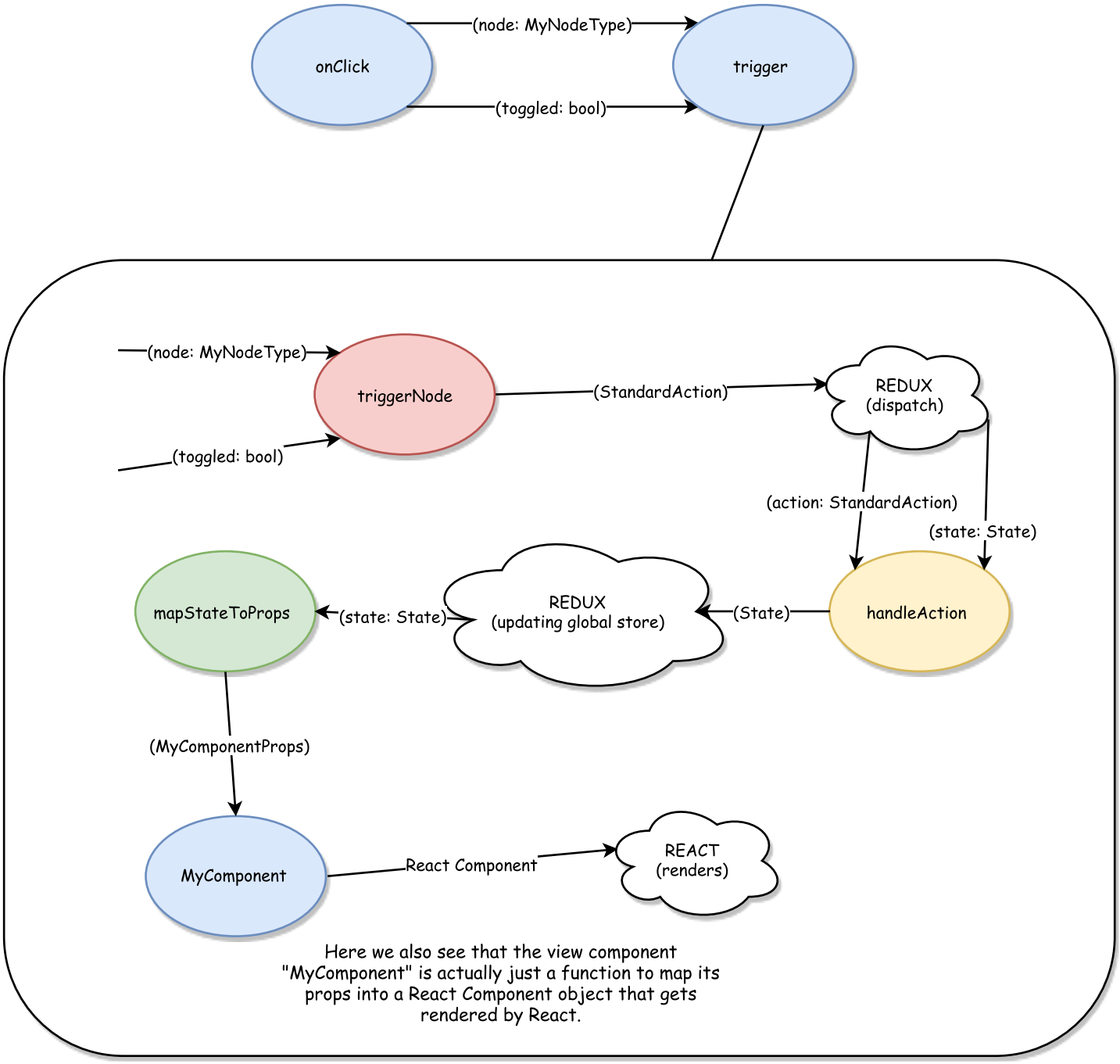


You may now think of Redux being "wired up" internally with the following function:



Now for the flow!

Suppose "trigger" is called from our component (e.g. by clicking on a part in our component that also delivers the two arguments node and bool)



Here we also see that the view component "MyComponent" is actually just a function to map its props into a React Component object that gets rendered by React.